МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

Дніпровський національний університет залізничного транспорту імені академіка В. Лазаряна

Кафедра Комп'ютерні інформаційні технології

«ДО ЗАХИСТУ»

Завідувач кафедри

_____ /В. І. Шинкаренко/

«_18_» _грудня_ 20_20_р.

ДИПЛОМНА РОБОТА

на здобуття освітнього ступеня «магістр»

Галузь знань **12 Інформаційні технології**

Спеціальність **121 Інженерія програмного забезпечення**

Тема **Дослідження часової ефективності алгоритмів комп'ютерної графіки для видалення невидимих ліній та поверхонь**

Theme **Research of the time efficiency of computer graphics algorithms for removing hidden lines and surfaces**

Керівник дипломної роботи доц. _____ О. П. Іванов

Нормоконтролер доц. _____ О. С. Куроп'ятник

Студент групи П31921 _____ І. О. Строчіков

Student Strochikov Ihor

Дніпро – 2020

Дніпровський національний університет залізничного транспорту імені академіка В. Лазаряна

Факультет Комп'ютерних технологій і систем кафедра Комп'ютерні інформаційні технології

Спеціальність Інженерія програмного забезпечення

«ЗАТВЕРДЖУЮ»
Завідувач кафедри
_____ проф. Шинкаренко В.І.
(підпис)
«12» грудня 2020 р.

ЗАВДАННЯ

до дипломної роботи на здобуття ОС Магістр
(освітній ступень)

студента групи (ПЗ1921) 961-М Строчікова Ігоря Олександровича
(номер групи) (ПІБ)

1 Тема дипломної роботи: Дослідження часової ефективності алгоритмів комп'ютерної графіки для видалення невидимих ліній та поверхонь затверджена наказом по університету від «10» жовтня 2019 р. № 779ст.

2 Термін подання студентом закінченого проекту «21» грудня 2020 р.

3 Вихідні дані до дипломного проекту_____

_____

_____

4 Зміст пояснювальної записки (перелік питань до розробки) проведення аналізу часової ефективності алгоритмів видалення невидимих ліній та поверхонь, визначення актуальних проблем, розробка власного алгоритму.

5 Перелік демонстраційного матеріалу презентація дослідження часової ефективності алгоритмів видалення невидимих ліній та поверхонь, результати експериментів, висновки; відео демонстрація роботи розробленого програмного інструментарію для проведення досліджень.

6. Консультанти (з назвами розділів):

Розділ	Консультант	Підпис, дата	
		завдання видав	завдання прийняв
Техніко-економічні розрахунки	доц. Гненний М.В.	03.11.20	10.12.20
Охорона праці та безпека в надзвичайних ситуаціях	ст. викладач Музикін М.І.	23.11.20	10.12.20

КАЛЕНДАРНИЙ ПЛАН

№ п/п	Назва розділів дипломного проекту	Термін виконання розділів проекту (роботи)	Примітка
1	Вступ	01.09.2020 – 10.09.2020	
2	Огляд літератури	10.09.2020 – 15.09.2020	
3	Постановка задачі, технічне завдання	15.09.2020 – 30.09.2020	
4	Створення тестової програми	01.10.2020 – 15.10.2020	
5	Перші тестування	15.10.2020 – 22.10.2020	30%
6	Аналіз результатів	30.10.2020 – 11.11.2020	
7	Розрахунок економічних показників	11.11.2020 – 14.11.2020	
8	Охорона праці	14.11.2020 – 18.11.2020	60%
9	Оформлення пояснювальної записки	18.11.2020 – 01.12.2020	
10	Демонстраційні матеріали	05.11.2020 – 16.12.2020	100%

Дата видачі завдання «10» жовтня 2019 р.

Керівник дипломного проекту _____ (підпис) Іванов О.П. (ПІБ)

Завдання прийняв до виконання _____ (підпис) Строчіков І.О. (ПІБ)

РЕФЕРАТ

Об’єктом даного дослідження є алгоритми комп’ютерної графіки для видалення невидимих ліній та поверхонь.

Предметом дослідження є вплив роздільної здатності екрану, кількості полігонів об’єктів та кількості пікселів для обробки на часову ефективність алгоритмів видалення невидимих ліній та поверхонь.

Метою даної роботи є визначення рівня впливу зміни різноманітних характеристик на часову ефективність алгоритмів видалення невидимих ліній та поверхонь.

Методи дослідження: проведення експериментів, збір даних експериментів та аналіз зібраних даних.

Результати та їх новизна: результати дослідження можуть бути використані іншими при розробці нових або гібридних алгоритмів, які схожі на ті, що досліджуються.

Пояснювальна записка складається зі вступу, 5 розділів, висновків, бібліографічного списку та 4 додатків:

- у вступі описується сутність роботи та її актуальність (3 сторінки);
- у першому розділі висвітлено аналіз сучасного дослідження проблеми, а також проаналізовано сучасний стан програмно-апаратного забезпечення (14 сторінок);
- у другому розділі надано обґрунтування напрямку дослідження (6 сторінок);
- у третьому розділі описано процес проектування і розробки інструментального забезпечення для дослідження (22 сторінки);
- у четвертому розділі описано виконані дослідження часової ефективності алгоритмів (15 сторінок);
- у п’ятому розділі розкриті питання охорони та безпеки праці (8 сторінок);
- додатки – технічне завдання, робочий проект, стаття та тези.

Таблиць – 11 , рисунків – 22, бібліографія – 71 джерело.

Ключові слова: комп’ютерна графіка, алгоритми видалення невидимих ліній та поверхонь, HSR-алгоритм (hidden surface removal algorithm), тривимірна модель.

ЗМІСТ

ВСТУП

В наш час вже більшість людей так чи інакше стикаються з комп’ютерною графікою, бо у кожного є власний смартфон або персональний комп’ютер. Для комфортного використання програм для моделювання або навіть ігор з різноманітною графікою дуже важливо аби об’єкти на сцені відмальовувались максимально швидко з прийнятною частотою (40-60 кадрів в секунду). Це особливо важливо для мобільних пристроїв, які є дуже популярними зараз, але водночас ще дуже поступаються персональним комп’ютерам в обчислювальній потужності.

Тому варто звернути увагу на часову ефективність алгоритмів видалення невидимих ліній та поверхонь, бо вони займають доволі велику частку загального часу рендеру об’єктів. Як загально відомо – кожен алгоритм видалення невидимих ліній та поверхонь має свої переваги та недоліки, тому є сенс дослідити, які з них працюють ефективніше в тій чи іншій ситуації.

Ці алгоритми мають єдину ціль – якомога швидше вирішити, які з вхідних пікселів потрібно відобразити, а які – ні. Це робиться на основі даних про кожний піксель. В загальному випадку просто порівнюються так звані z-координати (дальність від погляду користувача) кожного пікселя один з одним. А кожний алгоритм – це конкретна реалізація цього порівняння, яка може опинитися більш або менш вдалою для деякої ситуації.

Задача, яку вирішують алгоритми видалення невидимих ліній та поверхонь відносяться до задач м’якого реального часу, бо для користувача хоч і не бажано, але припустимо, коли час вирішення задачі буде перевищувати допустимий. Це може трапитися в тому випадку, коли сцена, що відображається, є дуже складною і неоптимізованою для поточного пристрою для відображення. Такі випадки приводять до того, що частота рендеру сцени стає меншою, через що користувач може помітити сіпання на екрані або навіть повністю статичну сцену з рідкою зміною кадрів приблизно 1 у секунду. Через це глядач не зможе належним чином зануритися у процес і нормально реагувати на побачене.

Актуальність роботи. На сьогоднішній день існує безліч алгоритмів видалення невидимих ліній та поверхонь, але на практиці використовуються лише деякі з них.

Це ті алгоритми, які показують найкращі результати для більшості ситуацій. Наприклад, в широко відомій графічній бібліотеці OpenGL реалізовано лише алгоритм Z-буфера. І користувач бібліотеки зможе використати лише цей алгоритм для видалення невидимих ліній та поверхонь. І хоча цей алгоритм зможе опрацювати графічну сцену абсолютно будь-якої складності, це ще не означає, що він є найефективнішим в будь-якому сценарії.

До недавніх часів це не було серйозною проблемою, бо навіть якщо обраний алгоритм буде працювати трохи повільніше іншого, то різниця буде настільки малою, що користувач просто цього не помітить, а частота кадрів в секунду буде все ще прийнятною. Але зараз з'явилися мобільні гаджети, які мають набагато меншу обчислювальну потужність, аніж персональні комп'ютери. Тим паче вони мають меншу кількість оперативної пам'яті, що може позначитися на ефективності алгоритмів, яким для роботи якраз потрібно багато пам'яті.

З розширенням технічних можливостей пристроїв, виросли й вимоги до комп'ютерної графіки. Користувачі почали звикати до поточного стану, коли графіка, що відображається, виглядає дуже реалістичною за рахунок більш різноманітної палітри кольорів та плавної анімації. Але з появою мобільних пристроїв виявилося, що досягти такого ж результату й тут неможливо, тому розробникам мобільних додатків та ігор часто доводиться зменшувати якість комп'ютерної графіки або витрачати велику кількість часу та ресурсів на оптимізацію задля більшої продуктивності.

Об'єкт дослідження. Об'єктом дослідження є алгоритми комп'ютерної графіки для видалення невидимих ліній та поверхонь.

Предмет дослідження. Вплив роздільної здатності екрану, кількості полігонів об'єктів та кількості пікселів для обробки на часову ефективність алгоритмів видалення невидимих ліній та поверхонь.

Мета і завдання дослідження. Мета даної роботи полягає в тому, щоб визначити рівень впливу зміни різноманітних характеристик на часову ефективність алгоритмів видалення невидимих ліній та поверхонь. Також метою є визначення

числових показників впливу на час роботи алгоритму, за допомогою яких можна буде обчислювати очікуваний час роботи алгоритму.

Поставлена мета зумовлює необхідність вирішення наступного ряду завдань:

- розробка програмного інструментарію, за допомогою якого можна буде збирати інформацію з проведених експериментів у зручному для аналізу вигляді;
- дослідження та аналіз отриманих результатів;
- визначення числових показників впливу на час роботи алгоритму.

Методи дослідження. Проведення експериментів, збір даних експериментів та аналіз зібраних даних.

Наукова новизна. Удосконалено підхід до видалення невидимих ліній та поверхонь в тому, що можна обчислити очікуваний час роботи кожного з алгоритмів та обрати найефективніший для поточної ситуації алгоритм.

Практичне значення. Результати дослідження можуть бути використані іншими при розробці нових або гібридних алгоритмів видалення невидимих ліній та поверхонь, що схожі на ті, що досліджуються у роботі.

Апробація результатів дослідження. Процес та результати дослідницької роботи доповідались на семінарі кафедри КІТ 22.11.2020р., а також було опубліковано тези в журналі щорічної міжнародної науково-практичної конференції «Сучасні інформаційні та комунікаційні технології на транспорті, в промисловості та освіті», що проходила 15-16 грудня 2020 року.

Публікації за темою роботи. Підготовлено статтю «Дослідження часової ефективності алгоритмів видалення невидимих ліній та поверхонь у комп’ютерній графіці» до видання у фаховому журналі.

1 АНАЛІЗ ПРОБЛЕМИ ПРОДУКТИВНОСТІ АЛГОРИТМІВ ВИДАЛЕННЯ НЕВИДИМИХ ЛІНІЙ ТА ПОВЕРХОНЬ

1.1 Призначення та сфера застосування

У тривимірній комп'ютерній графіці визначення прихованої поверхні – це процес ідентифікації поверхонь та частин поверхні, які не можна побачити під певним кутом огляду. Алгоритм визначення прихованої поверхні – це рішення проблеми видимості, яка була однією з перших великих проблем у галузі тривимірної комп’ютерної графіки [1].

Алгоритми видалення невидимих ліній та поверхонь використовуються при відображенні будь-якої комп’ютерної графіки з тривимірними об’єктами. Вони застосовуються усіма сучасними редакторами для моделювання та графічними бібліотеками під час відображення тривимірних об’єктів на екрані.

1.2 Постановка задачі

Необхідно розробити програмний продукт, який можна буде застосувати для проведення експериментів з алгоритмами видалення невидимих ліній та поверхонь. Програма повинна давати можливість користувачеві обирати файл формату .stl, який містить координати точок тривимірної моделі для відображення, задати роздільну здатність зображення, алгоритм для обробки та кількість тестів у цьому експерименті.

В якості результату програма має повертати файл у форматі .csv з даними про результат експерименту. Він має містити такі показники:

- назва алгоритму, який було використано;
- роздільна здатність екрану, для якої було запущено експеримент;
- загальна кількість полігонів, яку мали об’єкти для відображення на екрані;
- об’єм часу, який було затрачено саме на час роботи алгоритму.

Програма має бути простою у використанні, не містити дуже складного та громіздкого інтерфейсу. Призначення всіх полів для вводу повинно бути простим та зрозумілим.

1.3 Огляд HSR-алгоритмів

Існує декілька алгоритмів видалення невидимих ліній та поверхонь, які можна поділити на дві основні групи [2]:

- алгоритми, працюючі у просторі об’єктів;
- алгоритми, працюючі у просторі зображення.

1.3.1 Порівняння алгоритмів, працюючих у просторах об’єктів та зображення

Алгоритми, які працюють у просторі об’єктів та які працюють у просторі зображення відрізняються не лише методом обробки. Специфіка кожного виду алгоритмів також впливає й на інші фактори, які можуть бути більш або менш важливими в деякій конкретній ситуації.

Наприклад, HSR-алгоритми, працюючі у просторі об’єктів є дуже точними. Ця точність може бути обмежена лише точністю обчислення. В той же час, алгоритми, працюючі у просторі зображення, дуже залежать від роздільної здатності екрану. Нижче наведена табл. 1.1, яка порівнює ці два способи обробки [3].

Таблиця 1.1 – Порівняння алгоритмів, працюючих у просторах об’єктів та зображення

	Працюючі у просторі об’єкту	Працюючі у просторі зображення
Система координат	Алгоритми працюють з фізичною системою координат, в якій описуються дані об’єкти	Алгоритми працюють з системою координат екрану, на якому відображаються об’єкти
Точність	Висока точність, яка може бути обмежена лише точністю обчислень	На точність обчислень впливає роздільна здатність екрану
Об’єм обчислень	Зростає теоретично, як квадрат числа об’єктів n^2	Зростає теоретично, як N * n, де n – кількість об’єктів на сцені, N – кількість пікселів
Ефективність	Менш ефективні	Більш ефективні

Як можна побачити з наведеної таблиці, загалом алгоритми, які працюють у просторі об’єкту є більш точними, але менш ефективними, ніж алгоритми, працюючі у просторі зображення.

Тому, коли йдеться мова про час виконання алгоритму, а не його точність, то слід звернути увагу саме на другу категорію алгоритмів. У більшості випадків користувач навіть не побачить деякі неточності у зображенні, враховуючи, що сучасні екрани мають досить велику роздільну здатність.

Зараз найчастіше користувачам потрібна саме продуктивність, а не точність, особливо при відображенні комп'ютерної графіки в іграх, де потрібно обчислювати велику кількість об'єктів на сцені з прийнятною частотою до 60 кадрів у секунду.

1.3.2 Огляд алгоритму Z-буфера

У комп'ютерній графіці z-буферизація, також відома як глибина буферизації, - це управління координатами глибини зображення в тривимірній графіці, як правило, виконується в апаратному забезпеченні, іноді в програмному забезпеченні. Це одне з рішень проблеми видимості, це проблема вирішення, які елементи візуалізованої сцени видно, а які приховано. Z-буферизація була вперше описана у 1974 році Вольфгангом Штрассером. Алгоритм Z-буфера є алгоритмом, працюючих у просторі зображення для виявлення прихованої поверхні. Z-буфер може посилатися на структуру даних або на метод, що використовується для виконання операцій над цією структурою [4].

У механізмі 3d-рендерінгу, коли об'єкт проектується на екран, глибина (значення z) сформованого пікселя в проектованому зображенні екрана зберігається в буфері (z-буфер або буфер глибини). Значення z - це міра перпендикулярної відстані від пікселя на площині проекції до відповідної йому 3D-координати на багатокутнику у світовому просторі.

Z-буфер має ту саму внутрішню структуру даних, що і зображення, а саме 2d-масив, з тією лише різницею, що він зберігає z-значення для кожного пікселя екрана замість піксельних даних. Він має ті самі розміри, що і буфер екрану, за винятком випадків, коли використовується кілька z-буферів, наприклад, при отриманні розділеного екрана. Він працює в просторі екрана і приймає за вхідний сигнал проектоване зображення, яке походить від проекції об'єкта на екран.

Перш ніж зробити проекцію із світу на екран, зазвичай проводяться первинні тести видимості (наприклад, вибракування задньої сторони). Перш ніж зображення

передається в z-буфер, вторинні тести видимості (такі як перевірка накладання та відсікання екрану) зазвичай виконуються на вершинах об'єктів. Первинні та вторинні тести видимості не вимагають перевірки окремих пікселів, тому z-буфер звільняється від деяких обов'язків.

Під час перегляду зображення, що містить частково або повністю перекриваються непрозорі об'єкти або поверхні, неможливо повністю побачити ті об'єкти, які знаходяться найдальше від оглядача та за іншими об'єктами (тобто деякі поверхні приховані за іншими). Ідентифікація та видалення цих поверхонь називається проблемою прихованої поверхні. Щоб зменшити час візуалізації, приховані поверхні слід видалити перед передачею проектованого зображення поверхонь у z-буфер. Щоб перевірити накладання, z-буфер обчислює значення z пікселя, що відповідає першому об'єкту, і порівнює його зі значенням z в тому ж місці пікселя в z-буфері, що відповідає об'єкту, який, як відомо найближче до глядача. Якщо обчислене значення z менше, ніж значення z, яке вже є в z-буфері, тоді поточне z-значення в z-буфері замінюється обчисленим значенням. Це не обов'язково означає, що перший об'єкт в цілому ближче до глядача, ніж найближчий відомий об'єкт, але, безумовно, означає, що відповідна 3d-точка z-значення на поверхні першого об'єкта у світовому просторі знаходиться ближче до глядач. Іншими словами, об’єкти перетинаються, і принаймні якась частина першого об’єкта знаходиться ближче і, отже, видима для глядача. Зрештою, z-буфер дозволить правильно відтворити звичне сприйняття глибини: близький об'єкт ховає один далі. Це називається z-вибракування.

Гранулярність z-буфера має великий вплив на якість сцени: традиційний 16-розрядний z-буфер може призвести до артефактів (званих "z-боротьбою" або зшиванням), коли два об'єкти знаходяться дуже близько один до одного [5]. Приклад такого артефакту зображено на рис. 1.1. Більш сучасний 24-розрядний або 32-розрядний z-буфер поводиться набагато краще, хоча проблема не може бути повністю усунена без додаткових алгоритмів. 8-розрядний z-буфер майже не використовується, оскільки він має занадто малу точність.

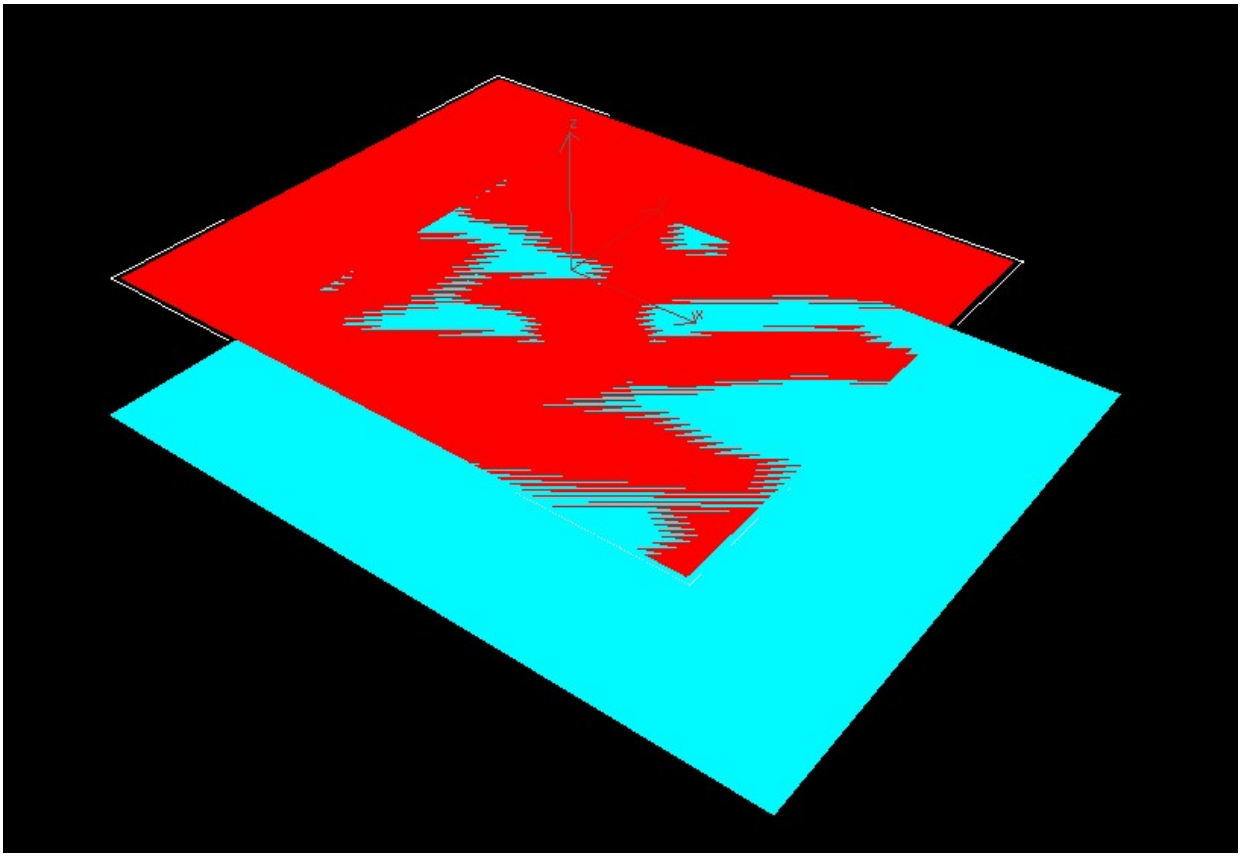

Рисунок 1.1 – Артефакт викликаний «Z-боротьбою»

1.3.3 Огляд алгоритму «художника»

Алгоритм художника (також алгоритм сортування по глибині та пріоритетного заповнення) – це алгоритм визначення видимої поверхні в тривимірній комп’ютерній графіці, який працює на основі полігона, а не піксель за пікселем, рядок за рядком, що є базисом інших алгоритмів видалення прихованої поверхні [6]. Алгоритм художника створює зображення шляхом сортування багатокутників у зображенні за їх глибиною та розміщення кожного багатокутника в порядку від найдальшого до найближчого об’єкта, як зображено на рис. 1.2.

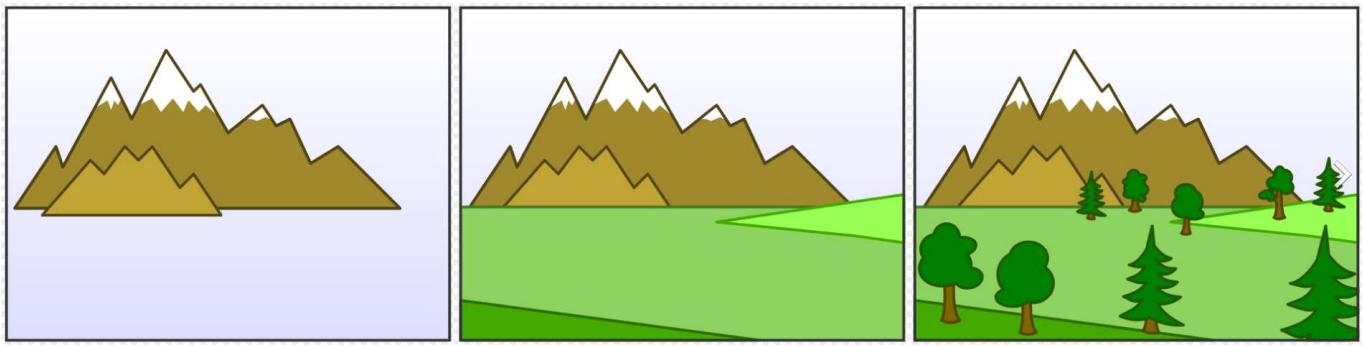

Рисунок 1.2 – Порядок роботи алгоритму «художника»

Алгоритм художника спочатку був запропонований як основний метод вирішення проблеми визначення прихованої поверхні Мартіном Ньюеллом, Річардом Ньюеллом і Томом Санчою в 1972 році. Назва "алгоритм художника" відноситься до техніки, що застосовується багатьма художниками, коли вони починають з малювання віддалених частин сцени перед частинами, які знаходяться ближче, тим самим охоплюючи деякі області віддалених частин. Подібним чином алгоритм художника сортує всі багатокутники сцени за їх глибиною, а потім малює їх у такому порядку, найбільш віддаленому від найближчого. Він буде зафарбовувати ті частини, які зазвичай не видно – таким чином вирішуючи проблему видимості – ціною малювання невидимих ділянок віддалених об'єктів. Впорядкування, що використовується алгоритмом, називається "глибинним порядком" і не повинно поважати числові відстані до частин сцени: суттєвою властивістю цього впорядкування є, швидше, те, що якщо один об'єкт перекриває частину іншого, перший об'єкт малюється після об'єкта, який він перекриває. Таким чином, дійсне впорядкування можна описати як топологічне впорядкування спрямованого ациклічного графіка, що представляє оклюзії між об'єктами.

Алгоритм «художника» має такі переваги:

- простота алгоритму (він є не таким складним порівняно з іншими алгоритмами, які сортують елементи по глибині);
- ефективність пам’яті (алгоритм надає пріоритет ефективному використанню пам'яті, але за рахунок більшої обчислювальної потужності, оскільки всі частини всіх зображень повинні бути відтворені [7]).

А також він має недоліки:

- у деяких випадках алгоритм може вийти з ладу, включаючи циклічне перекриття [8] (приклад зображено на рис. 1.3) або пересічення полігонів;
- не найкраща ефективність, бо потрібно відмалювати кожний полігон, навіть якщо він буде перекритий іншим полігоном.

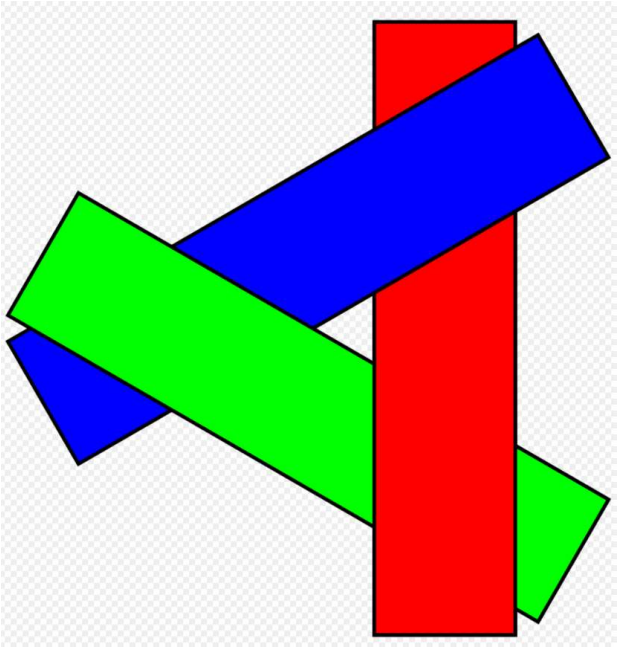

Рисунок 1.3 – Циклічне перекриття

Щоб вирішити проблему з циклічним перекриттям авторами алгоритму «художника» був запропонований алгоритм, який перевіряв такі випадки [9].

На фазі сортування по глибині, якщо два полігони не мають ступенів перекриття або граничних мінімальних та максимальних значень у напрямках x, y та z, їх можна легко сортувати. Якщо два багатокутники, Q і P, перекриваються у напрямку Z, то, можливо, необхідно відсікти деяку частину.

1.3.4 Огляд BSP-алгоритму

Розділення двійкового простору (binary space partitioning) – це загальний процес рекурсивного поділу сцени на дві, доки розбиття не задовольнить одну або кілька вимог. Це можна розглядати як узагальнення інших просторових деревних структур, таких як k-d дерева і чотирикутні дерева, наприклад, де гіперплощини, що розділяють простір, можуть мати будь-яку орієнтацію, замість того, щоб бути суміщеними з осями координат, як вони є у k-d деревах або квадратичних деревах. При використанні в комп’ютерній графіці для візуалізації сцен, що складаються з площинних багатокутників, площини перегородки часто вибирають так, щоб вони збігалися з площинами, визначеними полігонами в сцені [10]. Приклад процесу такого розбиття наведено на рис. 1.4.

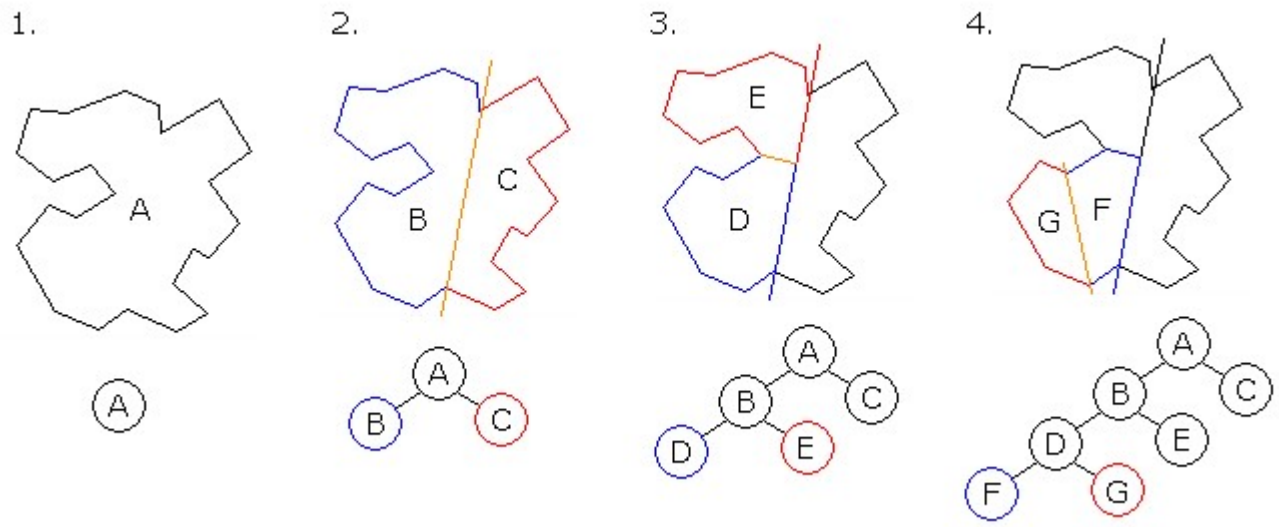

Рисунок 1.4 – Приклад розбиття зображення

Конкретний вибір площини секціонування та критерій для завершення процесу секціонування варіюється залежно від призначення BSP-дерева. Наприклад, при візуалізації комп’ютерної графіки сцена поділяється до тих пір, поки кожен вузол BSP-дерева не містить лише багатокутників, які можуть бути відтворені у довільному порядку. Отже, коли використовується відсікання задньої сторони, кожен вузол містить опуклий набір багатокутників, тоді як при відтворенні двосторонніх багатокутників кожен вузол BSP-дерева містить лише полігони в одній площині. При виявленні зіткнень або трасування променів сцену можна розділити на примітиви, на яких тести на зіткнення або перетин перекриття є простими.

Розділення двійкового простору виникло завдяки комп'ютерній графіці, необхідній для швидкого малювання тривимірних сцен, що складаються з багатокутників. Простий спосіб намалювати такі сцени – це алгоритм «художника», який створює багатокутники в порядку відстані від глядача, назад спереду, зафарбовуючи фон і попередні багатокутники кожним ближчим об’єктом. Цей підхід має два недоліки: час, необхідний для сортування багатокутників у зворотному порядку, і можливість помилок у перекритті полігонів. Фукс та співавтори показали, що побудова BSP-дерева вирішила обидві ці проблеми, забезпечивши швидкий метод сортування багатокутників щодо даної точки зору (лінійний за кількістю багатокутників у сцені) та вирішивши проблему циклічного перекриття, яке може виникати з алгоритмом «художника». Недоліком розділення двійкового простору є

те, що створення BSP-дерева може зайняти багато часу. Як правило, він виконується один раз на статичній геометрії, як крок попереднього обчислення, перед рендерингом або іншими операціями в реальному часі на сцені. Витрати на побудову BSP-дерева ускладнюють та неефективне безпосереднє впровадження рухомих об'єктів у дерево.

BSP-дерева часто використовуються у 3D-відеоіграх, особливо шутерах від першої особи. Ігрові двигуни, що використовують BSP-дерева, включають двигуни Doom, Quake, GoldSrc та Source. У них BSP-дерева, що містять статичну геометрію сцени, часто використовуються разом із Z-буфером, щоб правильно об'єднати рухомі об'єкти, такі як двері та символи, на фонову сцену. Хоча двійкове розбиття простору забезпечує зручний спосіб зберігання та отримання просторової інформації про багатокутники у сцені, це не вирішує проблему визначення видимої поверхні.

1.3.5 Огляд scan-line алгоритму

Построковий алгоритм відноситься до алгоритмів, що працюють у просторі зображення. Цей метод має інформацію про глибину лише для однієї лінії сканування. Для того, щоб отримати одну лінію глибини сканування, необхідно згрупувати та обробити всі полігони, що перетинають дану лінію сканування, одночасно перед обробкою наступної лінії сканування. Для цього ведуться дві таблиці – таблиця ребер та таблиця багатокутників [11].

Таблиця ребер – вона містить координатні кінцеві точки кожного рядка в сцені, зворотний нахил кожного рядка та вказівники на таблицю багатокутників для з'єднання країв з поверхнями.

Таблиця багатокутників – вона містить коефіцієнти площини, властивості матеріалу поверхні, інші дані про поверхню та може бути вказівниками на таблицю ребер.

Для полегшення пошуку поверхонь, що перетинають дану лінію сканування, формується активний список ребер. Активний список зберігає лише ті ребра, які перетинають лінію сканування в порядку збільшення X. Також для кожної поверхні встановлюється прапорець, який вказує, чи знаходиться позиція вздовж лінії сканування всередині або зовні поверхні.

Позиції пікселів на кожній лінії сканування обробляються зліва направо. На лівому перетині з поверхнею увімкнено поверхневий прапор, а праворуч - прапор. Обчислення глибини необхідно виконувати лише тоді, коли прапори кількох поверхонь увімкнені в певному положенні лінії сканування.

1.3.6 Огляд алгоритму Варнока

Алгоритм Варнока вирішує проблему візуалізації ускладненого зображення шляхом рекурсивного поділу сцени до тих пір, поки не будуть отримані області, які є тривіальними для обчислення. Іншими словами, якщо сцена досить проста для ефективного обчислення, вона відображається; в іншому випадку вона ділиться на менші частини, які також перевіряються на простоту [12]. Приклад процесу розбиття зображено на рис. 1.5.

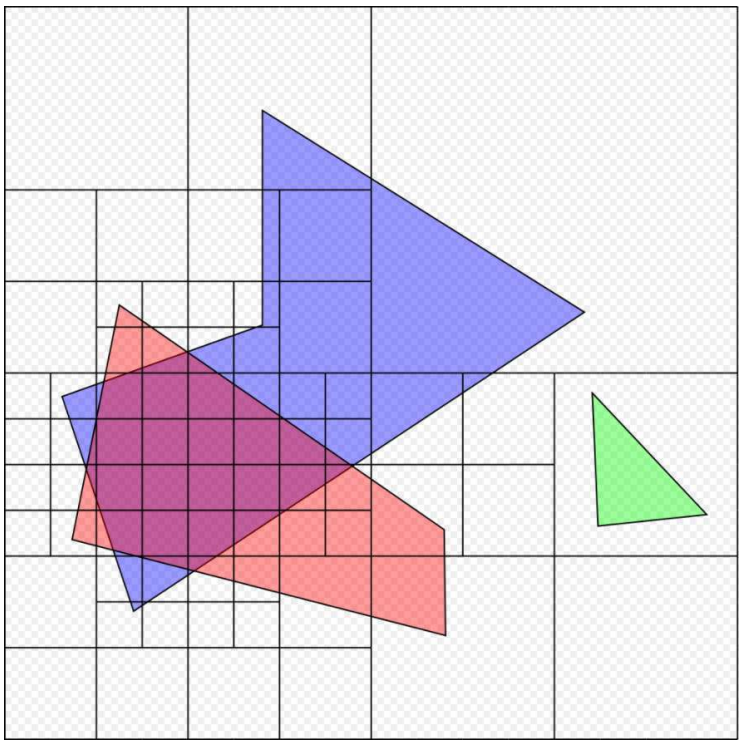

Рисунок 1.5 – Приклад розбиття зображення на частини

1.4 Огляд літератури

1.4.1 Растеризація

Растеризація – це завдання зробити зображення, описане у векторному графічному форматі (фігури), і перетворити його в растрове зображення (серію пікселів, крапок або ліній, які при спільному відображенні створюють зображення, яке було представлене за допомогою фігур) [13]. Потім растризоване зображення

може відображатися на дисплеї комп'ютера, відеодисплеї або принтері або зберігатися у форматі растрових файлів. Растеризація може стосуватися техніки малювання тривимірних моделей або перетворення 2D-примітивів візуалізації, таких як багатокутники, сегменти рядків, у растровий формат.

Растеризація – одна із типових технік візуалізації 3D-моделей. Порівняно з іншими методами візуалізації, такими як трасування променів, растеризація надзвичайно швидка і тому використовується в більшості 3D-движків реального часу. Однак растеризація – це просто процес обчислення відображення від геометрії сцени до пікселів і не передбачає конкретного способу обчислення кольору цих пікселів. Конкретний колір кожного пікселя призначається затіненням (що в сучасних графічних процесорах повністю програмоване). Затінення може базуватися на фізичних законах, їх наближеннях або суто художньому задумі.

Процес растеризації 3D-моделей на 2D-площині для відображення на екрані комп'ютера ("простір екрану") часто здійснюється за допомогою усталеного функціонального (непрограмованого) обладнання в графічному конвеєрі. Це пояснюється тим, що не існує мотивації для модифікації методів растеризації, що використовуються під час рендерингу, а спеціальна система забезпечує високу ефективність.

1.4.2 Полігональна сітка

У 3D-комп’ютерній графіці та моделюванні твердих тіл полігональна сітка – це сукупність вершин, ребер та граней, що визначає форму багатогранного об’єкта. Грані зазвичай складаються з трикутників (сітка трикутника), чотирикутників (чотирикутників) або інших простих опуклих багатокутників (n-кутників), оскільки це спрощує візуалізацію, але також може бути більш загально складеним із увігнутих багатокутників або навіть багатокутників з дірками [14].

Вивчення полігональних сіток – це велике підполе комп’ютерної графіки (зокрема, комп’ютерної графіки 3D) та геометричного моделювання. Різні подання полігональних сіток використовуються для різних застосувань та цілей. Різноманітність операцій, що виконуються над сітками, може включати: логічну логіку, згладжування, спрощення та багато інших. Існують також алгоритми

трасування променів, виявлення зіткнень та динаміки твердого тіла з полігональними сітками. Якщо краї сітки відображаються замість граней, тоді модель стає каркасною.

1.4.3 Формат файлу STL

Файл STL зберігає інформацію про тривимірні моделі. Цей формат описує лише геометрію поверхні тривимірного об'єкта без будь-якого подання кольору, текстури або інших типових атрибутів моделі [15].

Ці файли зазвичай генеруються програмою автоматизованого проектування (CAD) як кінцевий продукт процесу 3D-моделювання. “.STL” – це розширення файлу у форматі STL.

Формат файлу STL – це найбільш часто використовуваний формат файлу для 3D-друку. У поєднанні з 3D-різаком це дозволяє комп’ютеру взаємодіяти з апаратним забезпеченням 3D-принтера.

З самого скромного початку формат файлу STL був прийнятий і підтримувався багатьма іншими програмними пакетами САПР, і сьогодні широко використовується для швидкого створення прототипів, 3D-друку та автоматизованого виробництва.

Формат STL визначає як ASCII, так і двійкові подання. Бінарні файли є більш поширеними, оскільки вони є більш компактними [16].

Файл STL описує необроблену, неструктуровану тріангульовану поверхню за допомогою одиничної нормалі та вершин (упорядкованих за правилом правої сторони) трикутників за допомогою тривимірної декартової системи координат. В оригінальній специфікації всі координати STL мали бути позитивними числами, але це обмеження більше не застосовується, і негативні координати сьогодні часто зустрічаються у файлах STL. Файли STL не містять інформації про масштаб, а одиниці виміру є довільними [17].

1.4.4 Back-face culling метод

Back-face culling – це метод програмування комп'ютерної графіки, який визначає, чи видно багатокутник графічного об'єкта. Якщо полігон не видно, його відбирають від процесу візуалізації, що підвищує ефективність за рахунок зменшення кількості полігонів, які апаратне забезпечення має намалювати [18, 19].

Вершини фронтальних багатокутників звиваються за годинниковою стрілкою. Таким чином, багатокутники, які спрямовані подалі від камери, розташовані в порядку проти годинникової стрілки відносно поточного виду. Під час вибракування тильних граней ці багатокутники не малюються.

Цей метод може навіть добитися того ж результату, що й алгоритми видалення невидимих ліній та поверхонь в деяких випадках. Якщо все, що потрібно представити – це лише один опуклий багатогранник, то вибракування задньої поверхні еквівалентно HSR [19].

1.4.5 Програма для роботи з тривимірними моделями Autodesk MeshMixer

Autodesk MeshMixer - це безкоштовна програма для роботи з тривимірними сітковими моделями. У програмі немає інструментів для створення 3D-моделей, але є широкий набір інструментів для їх зміни і підготовки до тривимірного друку.

Користувач може імпортувати в MeshMixer будь-яку тривимірну модель, створену в будь-якому модельєрі, доопрацювати її, поліпшити і виправити, розмістити на віртуальному столі 3D-принтера і відправити на друк, або експортувати в формат STL. Сам MeshMixer працює з файлами * .MESH [20].

Деякі можливості цієї програми:

- просте Drag-and-Drop об'єднання мереж;
- точне позиціонування в 3D за допомогою маніпуляторів;
- конвертація в моделі для 3D-друку;
- просунуті інструменти вибору;
- згладжування і перетворення мереж;
- автоматичне вирівнювання поверхонь;
- 3D-патерни і гратчасті структури.

Висновки до розділу 1

В першому розділі були розглянуті існуючі алгоритми видалення невидимих ліній та поверхонь. В ході аналізу було виявлено недоліки та переваги кожного з них. Було з'ясовано, що на даний момент не існує універсального HSR-алгоритму, який ефективніше інших працює на всіх пристроях або у всіх випадках. Тому було

вирішено, що доцільно дослідити ефективність обраних алгоритмів в залежності від пристроїв та вхідних параметрів для того, щоб можна було розробити метод-перемикач, який буде використовувати найефективніший для конкретної ситуації HSR-алгоритм.

Для подальшого дослідження було обрано такі HSR-алгоритми:

- алгоритм Z-буферу;
- scan-line алгоритм;
- алгоритм «художника».

2 ОБҐРУНТУВАННЯ ЕКСПЕРИМЕНТАЛЬНОГО НАПРЯМКУ ДОСЛІДЖЕННЯ ЧАСОВОЇ ЕФЕКТИВНОСТІ АЛГОРИТМІВ ВИДАЛЕННЯ НЕВИДИМИХ ЛІНІЙ ТА ПОВЕРХОНЬ

Розробка нових та модифікація старих HSR-алгоритмів досі не втрачає своєї актуальності навіть не дивлячись на постійне зростання потужності ЕОМ. Зі зростанням потужності зростає й загальна складність сцен з тривимірними об’єктами в комп’ютерній графіці, які потрібно обробляти цим алгоритмам. Сучасні сцени стають складнішими через більшу деталізацію об’єктів, що потребує розбиття моделі на більшу кількість полігонів (найчастіше трикутників).

Крім того, зараз постійно зростає середня роздільна здатність екрану пристроїв. Алгоритмам потрібно обробляти все більшу кількість пікселів, що може погано вплинути на часову ефективність алгоритму. Якщо 20 років назад у більшості користувачів були пристрої з роздільною здатністю екранів 800x600, то зараз у середнього користувача вже є екран з роздільною здатністю 1920x1080 [21]. Тобто середня кількість пікселів для обробки змінилася з 480 тисяч до 2 мільйонів. І це не зважаючи на те, що вже зараз на ринку все частіше з’являються екрани з роздільною здатністю екрану UltraHD (3840x2160, 8.2 мільйони пікселів) [22], що може стати повсякденним через декілька років.

З роками вирішилася хіба що проблема алгоритмів, які потребували багато пам’яті. Наприклад, алгоритм Z-буферу потребує достатньо багато пам’яті, що було дуже великою проблемою для пристроїв того часу, коли цей алгоритм було розроблено. Ця проблема навіть підштовхнула до розробки алгоритму scan-line, який схожий на вищезгаданий алгоритм, але не потребував так багато пам’яті. Але зараз можна сказати, що проблема нестачі пам’яті відсутня, бо можливості ЕОМ в цьому плані дуже розширилися зрівняно з тими часами.

2.1 Поняття часової складності

В комп’ютерних науках складність часу – це обчислювальна складність, яка описує кількість комп’ютерного часу, необхідного для запуску алгоритму. Складність часу зазвичай оцінюється шляхом підрахунку кількості елементарних операцій, виконуваних алгоритмом, припускаючи, що кожна елементарна операція займає

фіксовану кількість часу для виконання. Таким чином, кількість зайнятого часу та кількість елементарних операцій, виконуваних алгоритмом, відрізняються щонайбільше на постійний коефіцієнт [23].

Оскільки тривалість роботи алгоритму може різнитися між різними входами одного і того ж розміру, зазвичай розглядається найгірша складність часу, тобто максимальна кількість часу, необхідна для входів даного розміру. Менш поширеною і, як правило, явно вказується, є складність середнього випадку, яка є середнім значенням часу, затраченого на входи певного розміру (це має сенс, оскільки існує лише кінцева кількість можливих входів даного розміру). В обох випадках складність часу, як правило, виражається як функція розміру вхідних даних.

Алгоритмічні складності класифікуються за типом функції, що фігурує у великих позначеннях O. Наприклад, алгоритм із часовою складністю O(n) – це лінійний часовий алгоритм та алгоритм із часовою складністю $O(n^a)$ для деякої константи $a > 1$ є поліноміальним часовим алгоритмом.

Але часова ефективність алгоритмів видалення невидимих ліній та поверхонь як правило залежить від декількох показників, а не лише одного. Майже всі HSR-алгоритми залежать хоча б від однієї з цих характеристик:

- роздільна здатність екрану;
- кількість полігонів для обробки;
- кількість пікселів для обробки.

Крім цього потрібно брати до уваги деяку конкретну реалізацію алгоритму, яка може бути не дуже вдалою, через що часова ефективність буде меншою. Також потрібно не забувати про мову програмування, на якій було реалізовано алгоритм, це теж може вплинути на ефективність алгоритму. І звісно ж найважливішим та найвпливовішим фактором є потужність пристрою, на якому цей алгоритм буде працювати.

2.2 Визначення коефіцієнта детермінації (R-квадрат)

Коефіцієнт детермінації – статистичний показник, що використовується в статистичних моделях як міра залежності варіації залежної змінної від варіації

незалежних змінних. Вказує наскільки отримані спостереження підтверджують модель [24].

Коефіцієнт детермінації визначається наступним чином:

$$R^2 = 1 - \frac{\sigma^2}{\sigma_y^2} \tag{2.1}$$

Для розрахунку вибіркового коефіцієнта детермінації, використовують вибіркові оцінки значень відповідних дисперсій:

$$R^2 = 1 - \frac{\hat{\sigma}^2}{\hat{\sigma}_y^2} = 1 - \frac{\frac{ESS}{n}}{\frac{TSS}{n}} = 1 - \frac{ESS}{TSS}, \tag{2.2}$$

де ESS – сума квадратів залишків регресії,

TSS – загальна сума квадратів.

Коефіцієнт детермінації в ході дослідження потрібен для того, щоб визначити наскільки отримані результати експериментів підтверджують модель. Також його визначення може допомогти у визначенні типу залежності. Наприклад, вона може бути лінійною, експоненціальною або логарифмічною. Визначити коефіцієнт детермінації можна легко за допомогою засобів Excel для побудованої лінії тренду.

2.3 Використання довірчого інтервалу

Довірчий інтервал – в математичній статистиці є типом інтервальної оцінки, яка обчислюється за даними спостереження, і покриває невідомий статистичний параметр із заданою надійністю. Це інтервал, у межах якого з заданою довірчою імовірністю можна чекати значення оцінюваної (шуканої) випадкової величини. Застосовується для повнішої оцінки точності в порівнянні з точковою оцінкою [25].

В ході дослідження може скластися така ситуація, коли деякі результати тестів будуть дуже відрізнятися від середніх. Це може бути викликано тим, що пристрій, на якому проводиться експеримент, був занятий деякою службовою задачею, яка й вплинула на час виконання алгоритму. Такі результати варто виключати з підрахунків, аби вони не впливали на загальну статистику, на основі якої будуть робитися деякі висновки.

Для цього й потрібні довірчі інтервали, аби виключати аномальні результати, які знаходяться за межею деякого заздалегідь визначеного довірчого інтервалу (наприклад, 90%).

2.4 Обчислення часової ефективності алгоритму

Часова ефективність кожного з досліджуваних алгоритмів буде обчислюватися за результатами експериментів. Дуже важливо виконати експеримент декілька сотень разів, щоб більш точно встановити середній показник часу. Середній показник часу для одного експерименту буде обчислюватися за такою формулою:

$$T_{avg} = \frac{\sum_{i=0}^{n} t_i}{n}, \tag{2.3}$$

де T_{avg} – середній час роботи алгоритму при заданих вхідних даних,

n – кількість експериментів,

t_i – час роботи алгоритму в і-тому експерименті.

Слід зауважити, що аномальні результати експериментів будуть виключені з загального підрахунку.

Після того, як будуть знайдені показники середнього часу для декількох різних експериментів, по цим даним буде побудований графік та лінія тренду до нього. За допомогою засобів програми Excel можна буде знайти функцію побудованої лінії тренду, наприклад, такого виду: y = 0.7*x + 53.2, якщо залежність буде лінійною. Підставляючи деяку характеристику (наприклад, роздільну здатність екрану) під x, можна буде обчислити приблизний час роботи алгоритму.

На основі обчисленого приблизного часу роботи HSR-алгоритму можна буде вирішити, який з них краще використати у конкретній ситуації.

2.5 Вибір моделей для дослідження

Для дослідження часової ефективності алгоритмів видалення невидимих ліній та поверхонь були обрані різні тривимірні моделі для кожного з експериментів. Для одного з експериментів було вирішено використовувати модель кулі з різним ступенем деталізації. Це можна побачити на рис. 2.1. Мінімальний рівень деталізації – 80 полігонів, максимальний – 39 тисяч полігонів.

Фактично це одна й та сама модель, тому кількість пікселів для обробки буде зростати незначно зі зростанням кількості полігонів. Це необхідно для того, щоб виявити залежність часової ефективності алгоритму саме від кількості полігонів.

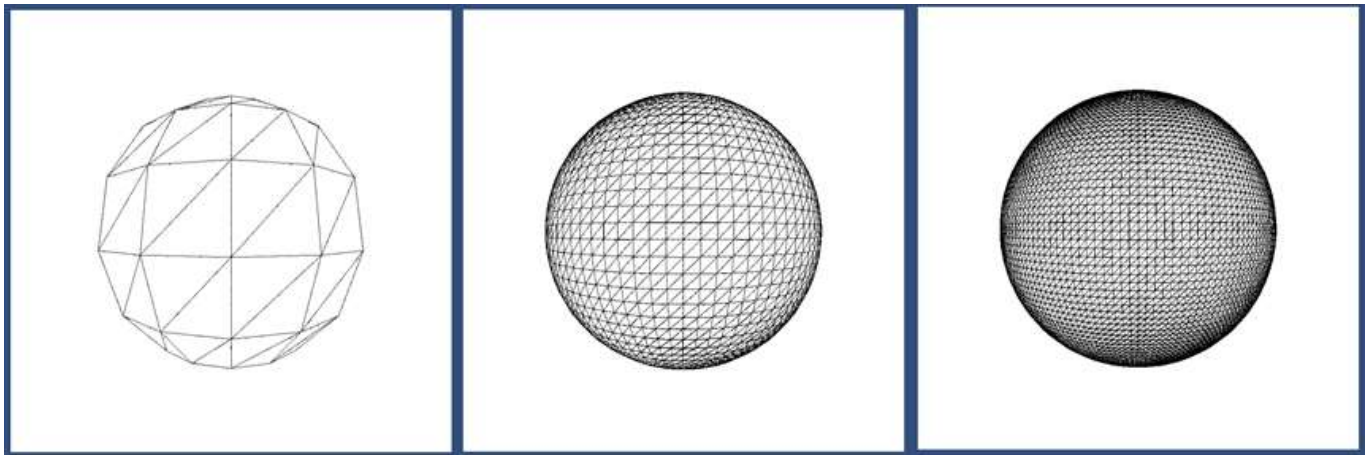

Рисунок 2.1 – Модель кулі з різним степенем деталізації

Для іншого експерименту була обрана дуже проста модель площини, яка містить всього 8 полігонів. Це було зроблено для того, щоб виключити залежність алгоритму від кількості полігонів моделі. Зображення цієї моделі на рис. 2.2.

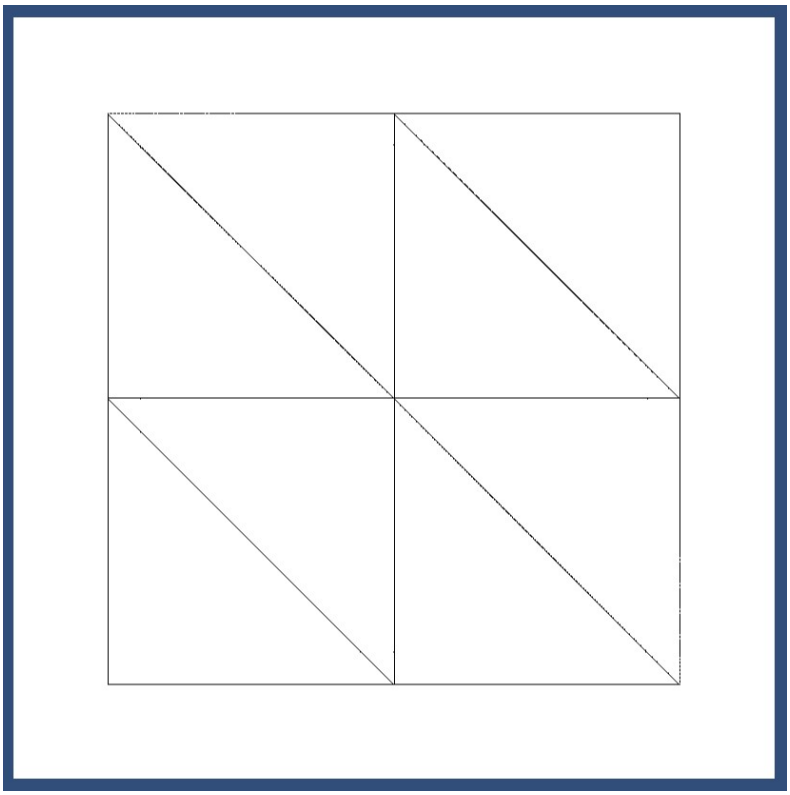

Рисунок 2.2 – Модель площини з 8 полігонами

Для останнього експерименту було обрано модель чайника середньої складності з 10 тисячами полігонів. Він буде використаний для експерименту з дослідження залежності часової ефективності алгоритму від роздільної здатності екрану. Зображення цієї моделі на рис. 2.3.

Рисунок 2.3 – Тривимірна модель чайника

Висновки до розділу 2

В другому розділі було освітлено та обґрунтовано експериментальний напрямок дослідження часової ефективності алгоритмів видалення невидимих ліній на поверхонь. У розділі було визначено, що часова ефективність досліджуваних алгоритмів буде оцінюватися наступним чином:

1) проведення деякого експерименту з конкретним алгоритмом безліч разів;
2) виключення аномальних результатів, які знаходяться за межею деякого довірчого інтервалу;
3) повторити шаг 1 і 2, але з іншими вхідними даними;
4) побудувати графік на основі отриманих результатів;
5) побудувати лінію тренду до цього графіку;
6) визначити функцію побудованої лінії тренду.

Також було визначено, які тривимірні моделі будуть використані для проведення експериментів.

3 ПРОЕКТУВАННЯ ТА РОЗРОБКА ПРОГРАМНОГО ІНСТРУМЕНТАРІЮ ДЛЯ ДОСЛІДЖЕННЯ ЧАСОВОЇ ЕФЕКТИВНОСТІ HSR-АЛГОРИТМІВ

3.1 Зовнішнє проектування

3.1.1 Вхідні данні

Вхідними даними програми є:

- ширина та висота зображення (цілі числа);
- кількість тестів (ціле число);
- унікальний номер алгоритму для використання (ціле число);
- файл формату .STL, який потрібно обробити (текст).

3.1.2 Вихідні дані

Результатом роботи програми є наступні вихідні дані:

- візуальне відображення обраної тривимірної моделі;
- файл формату .CSV, який містить результати експерименту та має такі дані:
  a. використаний HSR-алгоритм;
  b. роздільна здатність екрану;
  c. кількість полігонів моделі;
  d. час роботи алгоритму.

3.1.3 Формалізація задач

Формалізація задачі на рівні зовнішнього проектування представлена у вигляді діаграми варіантів використання.

Користувач представлений у вигляді актора, що взаємодіє з системою за допомогою варіантів використання. Варіанти використання надають опис можливостей, які система надає акторам.

На діаграмах використані наступні типи відношень між варіантами використання та акторами [26]:

- відношення асоціації – відображає зв’язок між акторами та варіантом використання. Відображається лінією зі стрілкою між акторами і варіантом використання;

- відношення включення – показує, що варіант використання включається в базову послідовність, позначається стрілкою з поміткою «include».

Користувач може виконувати наступні варіанти використання:

- встановити необхідні параметри експерименту;
- запустити тестування алгоритму;
- повернутися до вікна встановлення параметрів експерименту;
- зберегти файл з результатами експерименту.

Схема варіантів використання (Use-case diagram) наведена на рис. 3.1:

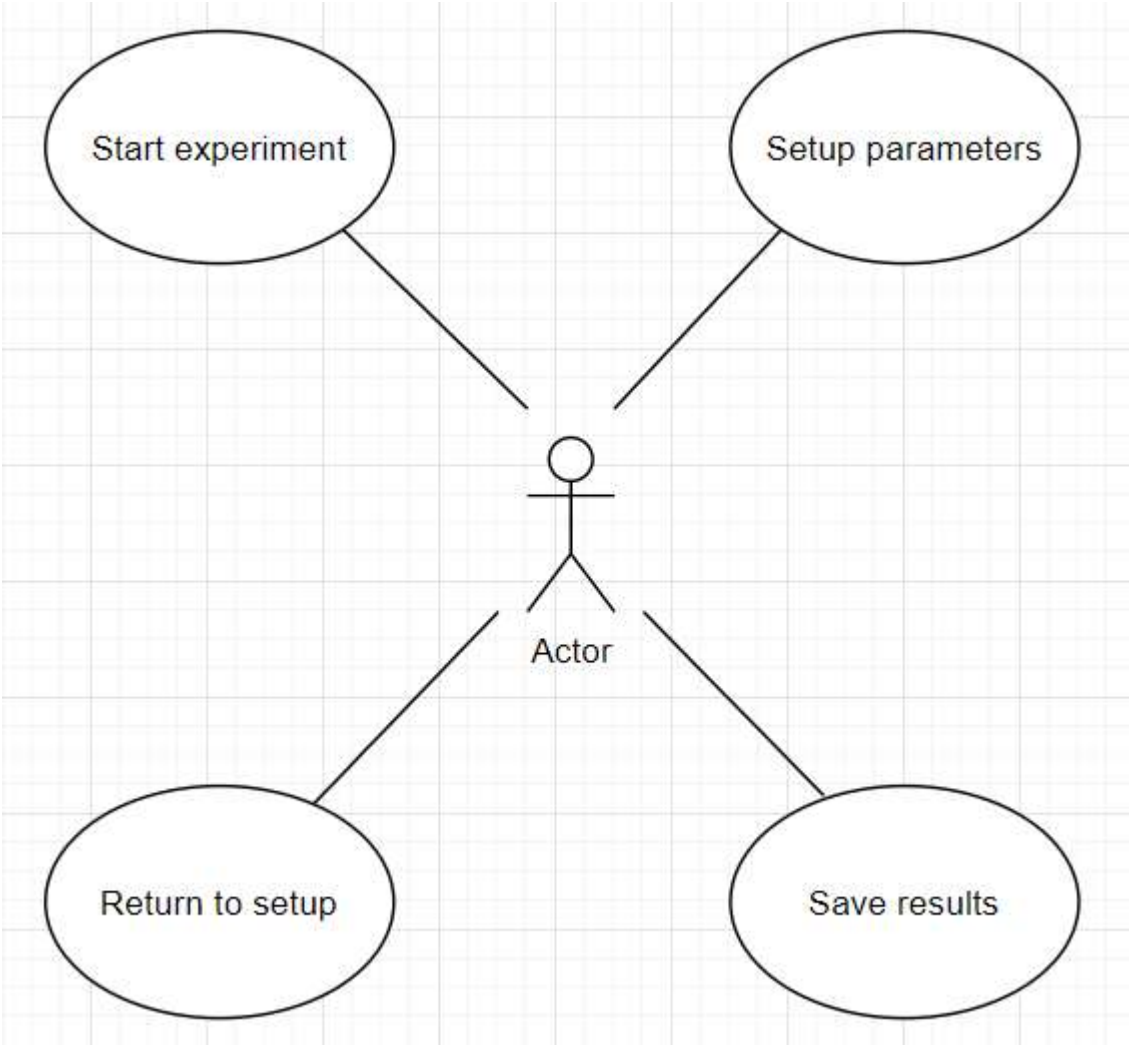

Рисунок 3.1 – Схема варіантів використання

3.1.4 Проектування динаміки системи

Перш за все потрібно визначити, які можуть бути сценарії використання системи, що розроблюється. Для цього можна побудувати UML діаграму послідовності.

Діаграма послідовності (sequence diagram) показує взаємодію об’єктів, розміщену у часовій послідовності [31]. Вона зображує об'єкти, що беруть участь у сценарії, та послідовність повідомлень, якими обмінюються об'єкти, необхідні для здійснення функціональних можливостей сценарію. Діаграми послідовності, як правило, асоціюються з реалізацією випадків використання в логічному поданні системи, що розробляється.

Діаграма послідовності показує у вигляді паралельних вертикальних ліній (життєвих ліній) різні процеси або об'єкти, що живуть одночасно, і, як горизонтальні стрілки, повідомлення, якими обмінюються між собою, у порядку, в якому вони відбуваються. Це дозволяє конкретизувати прості сценарії виконання графічно.

На рис. 3.2 зображено розроблену діаграму послідовності для одного основного сценарію роботи користувача з системою. Актор, тобто користувач, є ініціатором послідовності дій.

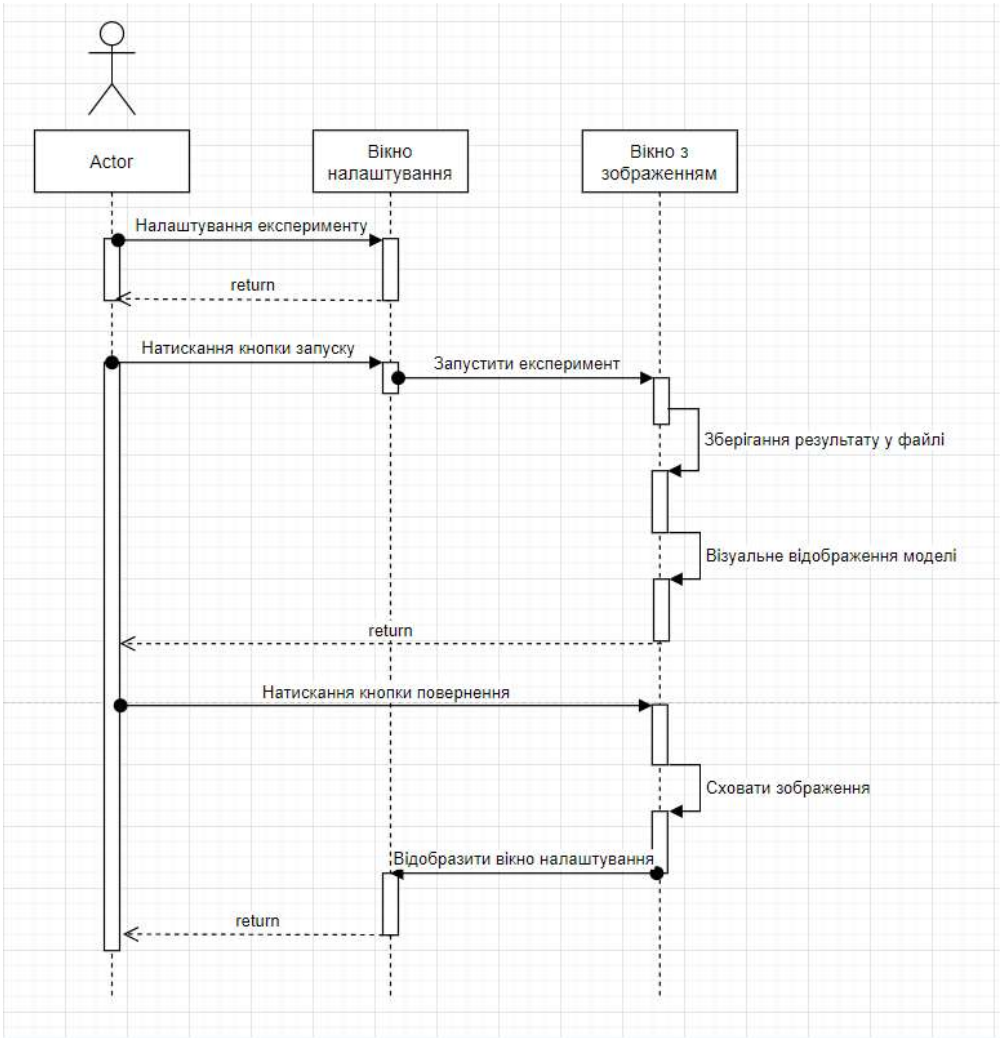

Рисунок 3.2 – Діаграма послідовності

3.1.5 Проектування інтерфейсу користувача

Проектування програмного інтерфейсу користувача грає важливу роль в етапі розробки програми, оскільки невдало спроектований інтерфейс ускладнить роботу з нею і може викликати у користувача небажання працювати із програмою.

Оскільки програмний продукт націлений на користувачів, що можуть не мати досвіду у роботі з подібними програмами, при розробці інтерфейсу необхідно дотримуватись цілого ряду правил:

- природність інтерфейсу. Не бажано змінювати звичні шляхи вирішення виробничої задачі та термінологію, що використовується у даній сфері діяльності [27];
- простота інтерфейсу. Інтерфейс повинен забезпечувати легкість у його вивченні та у використанні. Всі команди, повідомлення мають бути небагатослівними [28]. Треба розміщувати та представляти елементи на екрані з урахуванням їх смислового значення та логічного взаємозв’язку;
- естетична привабливість. Проектування візуальних компонентів є найважливішою складовою;
- заголовки повинні знаходиться біля або зверху редагованих полів;
- стандартні елемента, такі як кнопки «ОК» та «Скасувати» повинні розташовуватися згідно з замовчуваннями операційної системи;
- діалогові вікна повинні відповідати потоку даних між користувачем і системою;
- треба пам’ятати про надсилання підтверджень користувачеві. У випадку об’ємних команд користувач повинен отримувати інформацію про відправлення йому команди;
- у системи повинна бути проста обробка помилок [30]. Помилка повинна бути показана, а правильні дані повинні використовуватися для виконання наступного завдання;
- у системи повинна бути операція “відміна” [30]. У найпростішому випадку система повертається до останньої операції. У складніших – до попередніх;

– зв’язані операції повинні бути об’єднані в один діалог. Якщо це неможливо, операції повинні бути розділені таким чином, щоб зв’язані діалоги були доступні;

– потрібно дотримуватися правила Міллера. Правило Міллера говорить, що людина може зосередитися на 5-9 елементах [29]. Правило повинне застосовуватися при проектуванні меню, підменю, діалогових полів і т. д. Правило може бути реалізоване шляхом декомпозиції інтерфейсу і його подальшим угрупуванням в об’єднані групи.

Принцип обліку знань користувача припускає наступне: інтерфейс повинен бути настільки зручний при реалізації, щоб користувачам не знадобилося особливих зусиль, щоб звикнути до нього. У інтерфейсі повинні використовуватися терміни, зрозумілі користувачеві, а об'єкти, керовані системою, повинні бути безпосередньо пов'язані з робочим середовищем користувача.

Наприклад, якщо розробляється система, призначена для авіадиспетчерів, то керованими об'єктами в ній повинні бути літаки, траєкторії польотів, сигнальні знаки і тому подібне. Основну реалізацію інтерфейсу в термінах файлових структур і структур даних необхідно приховати від кінцевого користувача. Принцип узгодженості інтерфейсу користувача припускає, що команди і меню системи повинні бути одного формату, параметри повинні передаватися у всі команди однаково і пунктуація команд повинна бути схожою. Такі інтерфейси скорочують час на навчання користувачів. Знання, отримані при вивченні якої-небудь команди або частини додатку, можна потім застосувати при роботі з іншими частинами системи.

В даному випадку мова йде про узгодженості низького рівня. І творці інтерфейсу завжди повинні прагнути до нього. Проте бажана узгодженість є і більш високого рівня. Наприклад, зручно, коли для всіх типів об'єктів системи підтримуються однакові методи (такі, як друк, копіювання і тому подібне). Проте повна узгодженість неможлива і навіть небажана. Наприклад, операцію видалення об'єктів робочого столу доцільно реалізувати за допомогою їх перетягання в корзину. Але в текстовому редакторові такий спосіб видалення фрагментів тексту здається неприродним.

Завжди потрібно дотримуватися наступного принципу: кількість несподіванок повинна бути мінімальною, оскільки користувачів дратує, коли система раптом починає поводитися не передбачувано. При роботі з системою у користувачів формується певна модель її функціонування. Якщо його дія в одній ситуації викликає певну реакцію системи, природньо чекати, що така ж дія в іншій ситуації приведе до аналогічної реакції. Якщо ж відбувається зовсім не те, що очікувалося, користувач або дивується, або не знає, що робити. Тому розробники інтерфейсів повинні гарантувати, що схожі дії справлять схоже враження.

Дуже важливий принцип відновлюваності системи, оскільки користувачі завжди допускають помилки. Правильно спроектований інтерфейс може зменшити кількість помилок користувача (наприклад, використання меню дозволяє уникнути помилок, які виникають при введенні команд з клавіатури), проте всі помилки усунути неможливо. У інтерфейсах повинні бути засоби, що по можливості запобігають помилкам користувача, а також що дозволяють правильно відновити інформацію після помилок.

Підтвердження деструктивних дій – якщо користувач вибрав потенційно деструктивну операцію, то він повинен ще раз підтвердити свій намір.

Можливість відміни дій – відміна дії повертає систему в той стан, в якому вона знаходилася до їх виконання. Не зайвою буде підтримка багаторівневої відміни дій, оскільки користувачі не завжди відразу розуміють, що зробили помилку.

Наступний принцип – підтримка користувача. Засоби підтримки користувачів повинні бути вбудовані в інтерфейс і систему і забезпечувати різні рівні допомоги і довідкової інформації. Повинне бути декілька рівнів довідкової інформації – від основ для початкуючих до повного опису можливостей системи.

Довідкова система повинна бути структурованою і не перенавантажувати користувача зайвою інформацією при простих запитах до неї. Принцип обліку різнорідності користувачів припускає, що з системою можуть працювати різні їх типи. Частина користувачів працює з системою нерегулярно, час від часу.

Але існує і інший тип "досвідчені користувачі", які працюють з додатком щодня по декілька годин. Випадкові користувачі потребують такого інтерфейсу, який

"керував" би їх роботою з системою, тоді як досвідченим користувачам потрібний інтерфейс, який дозволив би їм максимально швидко взаємодіяти з системою.

Крім того, оскільки деякі користувачі можуть мати різні фізичні недоліки, в інтерфейсі повинні бути засоби, які допомогли б їм налаштувати інтерфейс під себе. Це можуть бути засоби, що дозволяють відображати збільшений текст, заміщати звук текстом, створювати кнопки великих розмірів і тому подібне.

3.1.6 Створення ескізів форм

На етапі проектування до остаточного затвердження інтерфейсу користувача зручно оперувати ескізами екранів. Під час розробки ескізів багато уваги уділялося внутрішньопрограмній узгодженості інтерфейсу. Назви управляючих кнопок, розташування ключових елементів форм і загальний стиль по можливості уніфікувалися або робилися схожими, з метою полегшення навчання користувачів при роботі з програмою.

Програма повинна мати простий та зрозумілий інтерфейс і складатись з таких екранів:

- екран налаштування експерименту (рис. 3.3);
- екран з відображеною тривимірною моделлю (рис. 3.4);

Розроблений інтерфейс є інтуїтивно зрозумілим для користувача, що зменшує ймовірність помилок при роботі.

Екран налаштування візуально розбитий на чотири секції, кожна з яких має своє призначення:

- секція Resolution;
- секція HSR Algorithm;
- секція Testing;
- секція Choose file.

Секція Resolution містить два поля для вводу. Перше поле відповідає за ширину зображення, а друге – за висоту. Ці поля мають текст Width та Height відповідно, який замінюється числами, які користувач може ввести. Це було зроблено для того, щоб користувач інтуїтивно розумів, яке поле відповідає за ширину, а яке – за висоту та випадково їх не сплутав.

Секція HSR Algorithm містить три перемикача (radio button). В залежності від обраного алгоритму, користувач може поставити флаг для одного з них. Слід замітити, що користувач може обрати лише один HSR-алгоритм. Якщо він спробує встановити флаг над іншим алгоритмом, то попередній флаг буде знято. Це було зроблено, щоб запобігти неоднозначності.

Секція Testing містить лише одне поле для вводу. Це поле відповідає за те, скільки разів буде запущено HSR-алгоритм. Користувач може ввести будь-яке число більше 0.

Остання секція Choose file містить лише одне спадне меню (dropdown). В закритому стані воно відображає файл, який на даний момент обрано для експерименту. Користувач може натиснути на нього, після чого меню розкриється та відобразить усі варіанти для вибору. Після цього користувач може обрати необхідну модель зі списку та натиснути на обраний варіант, після чого меню знову закриється, а відображатися буде вже щойно обраний варіант.

Екран налаштування має велику кнопку «Apply» знизу, натискання на яку запустить рендеринг обраної моделі на полотні обраного розміру у секції Resolution. Після цього екран налаштування зникне і з’явиться екран з відображеною тривимірною моделлю.

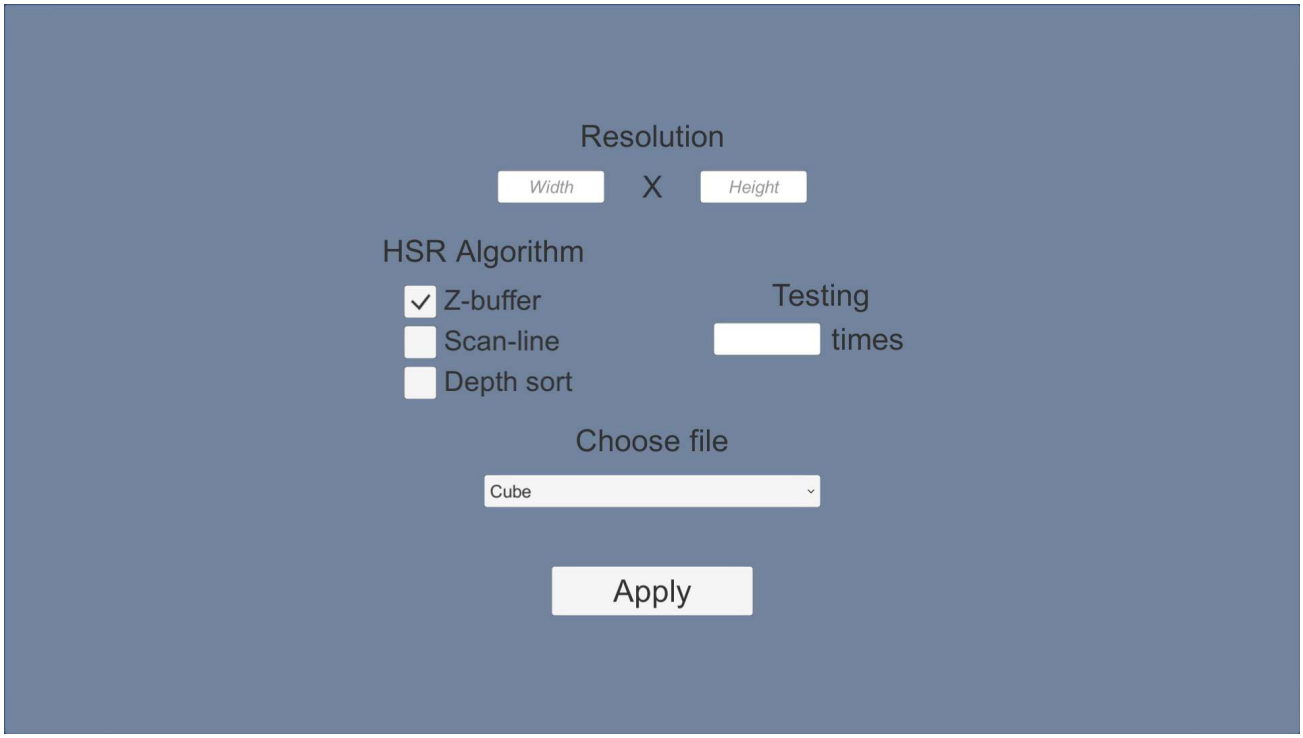

Рисунок 3.3 – Ескіз екрану налаштування експерименту

Екран з відображеною тривимірною моделлю має лише два елементи. Перший елемент – це полотно (canvas), розмір якого залежить від встановлених ширини та висоти на екрані налаштування експерименту. На цьому полотні відображається обрана раніше статична тривимірна модель. Відображена модель не може вийти за межі цього полотна. Другий елемент – це кнопка з назвою «Return». Натискання на цю кнопку повертає користувача до екрану налаштування експерименту, що й зрозуміло з короткої назви цієї кнопки. Після цього екран з відображеною тривимірною моделлю зникає.

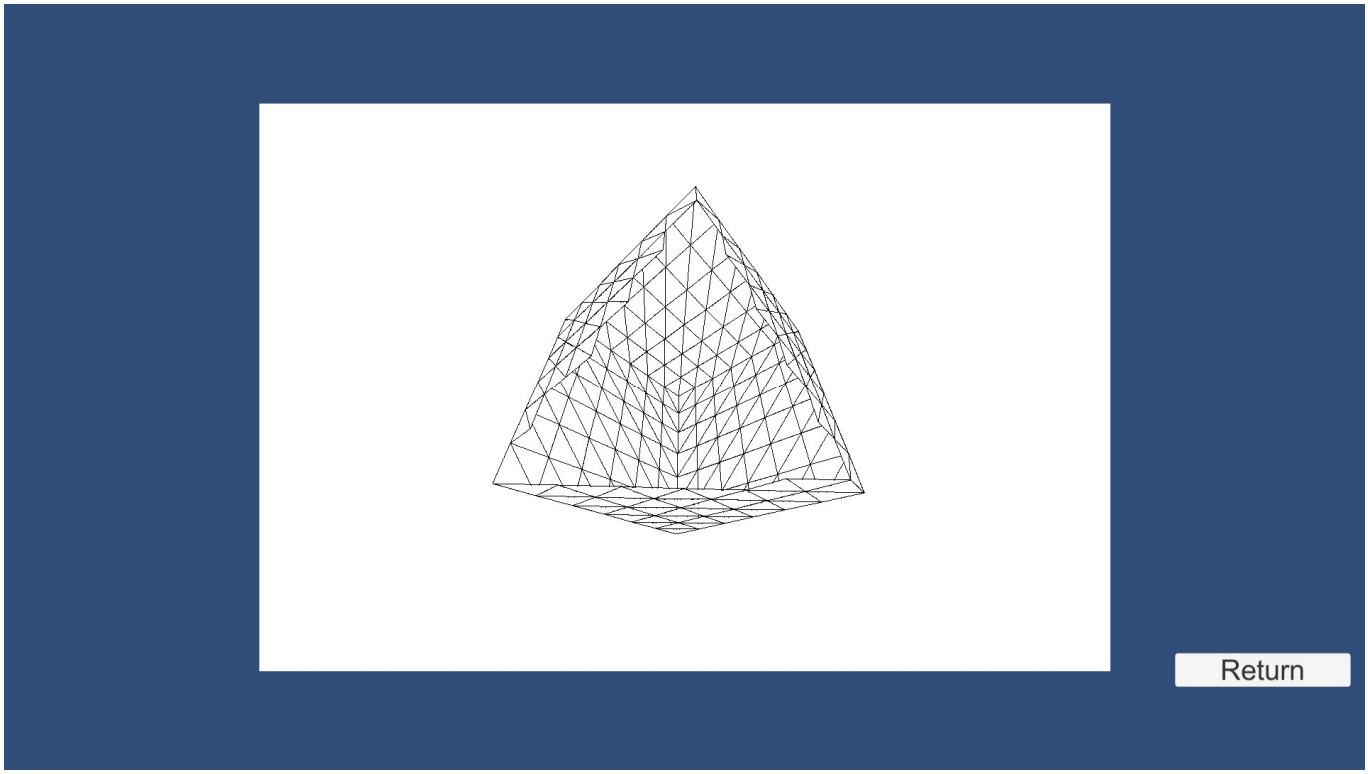

Рисунок 3.4 – Ескіз екрану з відображеною тривимірною моделлю

3.2 Зовнішнє проектування

3.2.1 Технологічна платформа

В якості технологічної платформи для розробки було використано ігровий движок Unity.

Unity – це крос-платформний ігровий движок, розроблений Unity Technologies, вперше оголошений і випущений у червні 2005 року на всесвітній конференції розробників Apple Inc. як ексклюзивний ігровий движок для Mac OS X. Станом на 2018 рік двигун був розширений для підтримки понад 25 платформ. Движок може бути використаний для створення тривимірних, двовимірних, ігор у віртуальній та

доповненій реальності, а також моделювання та іншого. Движок був прийнятий на виробництво за межами відеоігор, таких як кіно, автомобілебудування, архітектура, машинобудування та будівництво [32].

Як правило, ігровий движок надає безліч функціональних можливостей, що дозволяють їх задіяти в різних іграх, в які входять моделювання фізичних середовищ, карти нормалей, динамічні тіні і багато іншого. На відміну від багатьох ігрових движків, у Unity є дві основні переваги: наявність візуальної середовища розробки та мультиплатформена підтримка. Перший фактор включає не тільки інструментарій візуального моделювання, а й інтегровану середу, лінію складання, що направлено на підвищення продуктивності розробників, зокрема, етапів створення прототипів і тестування. Під мультиплатформенною підтримкою мається на увазі не тільки місця розгортання (установка на персональному комп'ютері, на мобільному пристрої, консолі і т. д.), але і наявність інструментарію розробки (інтегроване середовище може використовуватися під Windows і Mac OS).

Насамперед, було вирішено використовувати для розробки саме цей двигун з таких причин:

- безкоштовний;
- має безліч корисних вбудованих функцій, наприклад, функції підрахунку дистанції між двома точками у просторі.
- можна скомпілювати програму не лише на Windows, але й на Android та інші платформи, що дозволить проводити експерименти не лише на персональному комп'ютері.

3.2.2 Вибір мови програмування

Для розробки інструментарію для проведення досліджень було обрано мову C#.

C# – сучасна об'єктно-орієнтована і типобезопасна мова програмування [33]. C # відноситься до широко відомого сімейства мов C, і буде добре знайомою кожному, хто працював з C, C ++, Java або JavaScript.

C# надає мовні конструкції для безпосередньої підтримки такої концепції роботи. Завдяки цьому C # підходить для створення і застосування програмних

компонентів. З моменту створення мову C # збагатився функціями для підтримки нових робочих навантажень і сучасними рекомендаціями по розробці ПЗ.

Нижче представлені деякі з функцій мови, які забезпечують надійність та стійкість розроблюваних додатків:

- збірка сміття автоматично звільняє пам'ять, зайняту недосяжними невикористовуваними об'єктами [34];
- обробка винятків надає структурований і розширюваний підхід до виявлення помилок і відновлення після них [35];
- лямбда-вирази підтримують прийоми функціонального програмування [36];
- синтаксис запитів створює загальний шаблон для роботи з даними з будь-якого джерела [37];
- підтримка мов для асинхронних операцій надає синтаксис для створення розподілених систем [38];
- зіставлення шаблонів надає синтаксис для простого поділу даних з алгоритмів в сучасних розподілених системах [39];
- у C # діє єдина система типів [40].

3.2.3 Вибір системи контролю версіями

Система контролю версіями – програмне забезпечення для полегшення роботи зі змінною інформацією. Система управління версіями дозволяє зберігати кілька версій одного і того ж документа, при необхідності повертатися до попередніх версій, визначати, хто і коли зробив ту чи іншу зміну, і багато іншого [41].

Такі системи найбільш широко використовуються при розробці програмного забезпечення для зберігання вихідних кодів програми, що розробляється. Однак вони можуть з успіхом застосовуватися і в інших областях, в яких ведеться робота з великою кількістю безперервно змінюються електронних документів. Зокрема, системи управління версіями застосовуються в САПР, зазвичай в складі систем управління даними про виріб (PDM). Управління версіями використовується в інструментах конфігураційного управління.

В якості системи контролю версій було обрано Git. Git дозволяє мати кілька локальних гілок, які можуть бути повністю незалежними одна від одної [42].

Git швидкий, бо майже всі операції виконуються локально, що дає величезну перевагу в швидкості перед централізованими системами, яким постійно доводиться спілкуватися з сервером [43].

Git був побудований для роботи над ядром Linux, а це означає, що він повинен був ефективно обробляти великі сховища з першого дня. Git написаний на мові C, зменшуючи накладні витрати на виконання, пов'язані з мовами вищого рівня. Швидкість і продуктивність були головною метою дизайну Git з самого початку.

3.2.4 Взаємодія класів розроблюваної системи

Насамперед, потрібно визначитися з основним функціоналом системи, що буде розроблятися. Інструментарій для дослідження часової ефективності алгоритмів видалення невидимих ліній та поверхонь повинен мати такий функціонал:

- можливість задання параметрів експерименту (роздільна здатність екрану, кількість тестів, алгоритм для експерименту);
- парсинг файлів формату .STL для подальшої обробки;
- візуальне відображення тривимірної моделі з видаленими невидимими лініями та поверхнями;
- зберігання результатів часу роботи HSR-алгоритму з параметрами експерименту у файлі формату .CSV.

Оскільки часова ефективність буде досліджуватися для трьох HSR-алгоритмів, то є сенс додати клас-інтерфейс, який буде реалізовано трьома класами з логікою HSR-алгоритмів. Крім того, потрібен клас, який буде проводити обробку .STL файлу та зберігати зчитані координати полігонів у спеціальну структуру даних, яка теж буде представлена окремим класом-моделлю. Також потрібен клас, який буде відповідальний за відображення тривимірної моделі на полотні. І врешті-решт потрібен клас для зберігання результатів досліду у окремий файл формату .CSV.

Схема взаємодії класів зображена на рис. 3.5.

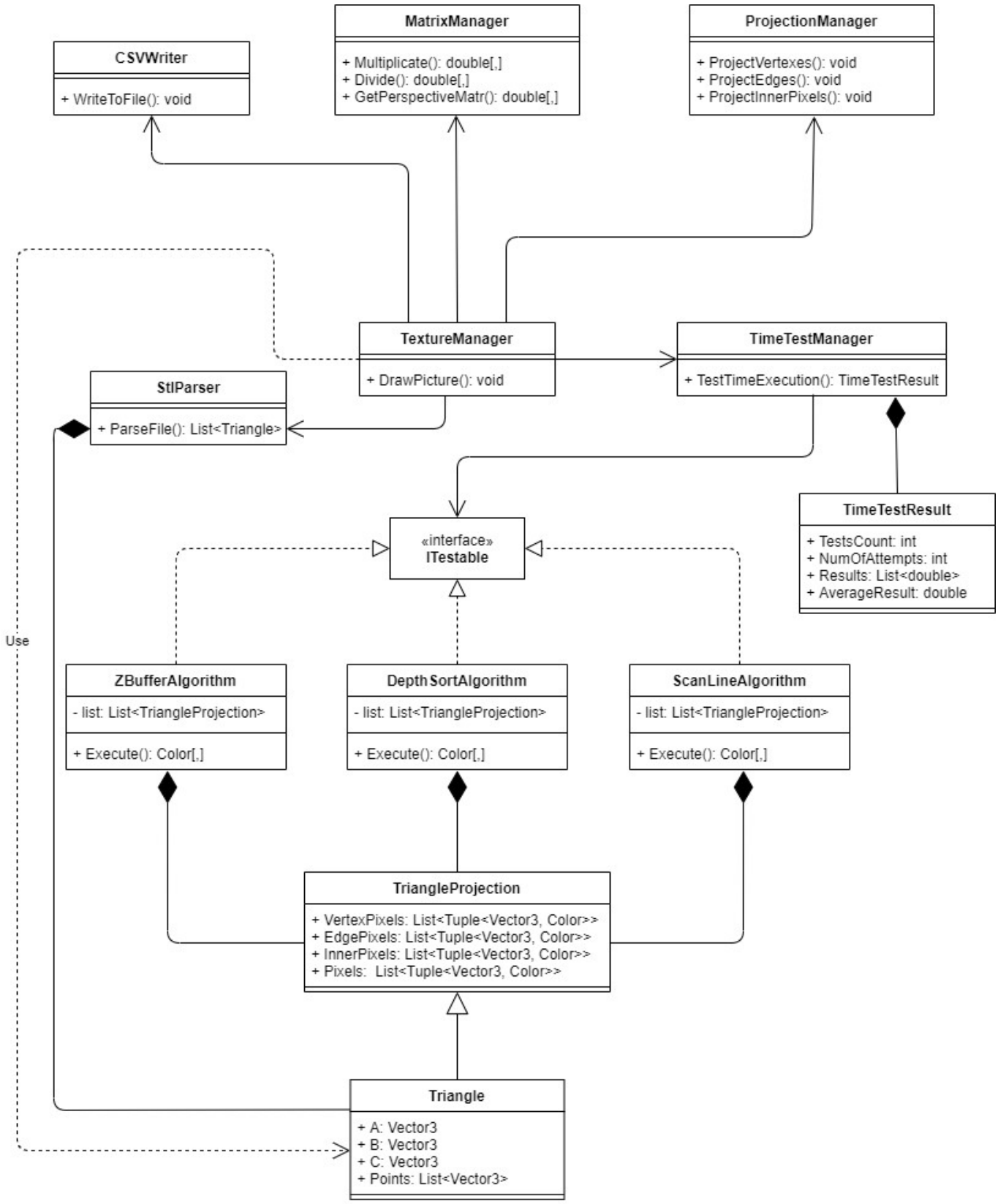

Рисунок 3.5 – Схема взаємодії класів

Розглянемо обов’язки кожного класу більш детально:

Клас TextureManager є головним класом у цій системі. Він по черзі викликає методи інших класів, щоб з назви отриманого файлу з моделлю можна було

відобразити вже оброблену модель на двовимірному полотні. Одразу він викликає метод ParseFile() класу StlParser та отримує список вершин кожного полігону з координатами у тривимірному просторі. Після цього він викликає методи класу MatrixManager та в результаті отримує матрицю для подальшого проектування. Потім по черзі викликаються методи ProjectVertexes(), ProjectEdges() та ProjectInnerPixels() класу ProjectionManager. Результатом цих викликів є список полігонів, які містять список пікселів вже з координатами для відображення в двовимірному просторі, координатою Z, яка означає дальність від погляду користувача, та кольором пікселя. Після цього викликається метод Execute() одного з обраних HSR-алгоритмів, який повертає двомірну матрицю з кольорами для відображення. Врешті-решт викликаються вбудовані функції Unity, щоб встановити колір пікселю на основі отриманої матриці.

Клас StlParser відповідає за парсинг файлів формату .STL. Слід зауважити, що він може парсити stl-файли лише формату ASCII. Використання бінарних stl-файлів не передбачено системою. Основний метод класу повертає список полігонів з трьома вершинами у тривимірному просторі.

Клас MatrixManager відповідає за арифметичні дії, які пов’язані з матрицями. Крім того, він має метод, який обчислює матрицю проектування по заданим параметрам.

Клас ProjectionManager відповідає за визначення, як саме повинні відображатися отримані полігони на двовимірному полотні. Він має метод ProjectVertexes(), який переводить вершини для відображення у двовимірному просторі. Також він має метод ProjectEdges(), який знаходить пікселі для відображення ребер полігону та отримує їх координати Z. Останній метод ProjectInnerPixels() знаходить всі пікселі для відображення, які знаходяться всередині полігонів, тобто тут не враховуються лише пікселі, які відносяться до ребер або вершин.

Клас Triangle – це клас-модель, який представляє собою полігон з трьома вершинами у тривимірному просторі. Ця модель використовується класом StlParser, коли він зберігає координати точок з файлу у цій моделі.

Клас TrianlgeProjection – це теж клас-модель, який є спадкоємцем класу Triangle. Це вже модель для представлення полігону у двовимірному просторі. Він використовується класом ProjectionManager та усіма класами, які реалізовують деякий HSR-алгоритм.

Інтерфейс ITestable – це інтерфейс, який реалізовується класами HSR-алгоритмів. Він потрібен для того, щоб можна було абстрагуватися від конкретної реалізації HSR-алгоритму. Зокрема, цей інтерфейс потрібен класу TimeTestManager, який викликає метод Execute() класу, який реалізує цей інтерфейс в даний момент.

Клас ZBufferAlgorithm – це клас, який реалізує інтерфейс ITestable та містить у собі метод Execute з логікою видалення невидимих ліній та поверхонь за допомогою алгоритму Z-буфера.

Клас ScanLineAlgorithm – це клас, який реалізує інтерфейс ITestable та містить у собі метод Execute з логікою видалення невидимих ліній та поверхонь за допомогою scan-line алгоритму.

Клас DepthSortAlgorithm – це клас, який реалізує інтерфейс ITestable та містить у собі метод Execute з логікою видалення невидимих ліній та поверхонь за допомогою алгоритму «художника».

Клас TimeTestManager відповідає за тестування обраного HSR-алгоритму. Його основний метод TestTimeExecution() приймає об’єкт інтерфейсу, який містить у собі об’єкт конкретної реалізації HSR-алгоритму. Цей метод запускає метод Execute() задану кількість разів та повертає об’єкт класу TimeTestResult.

Клас TimeTestResult – це клас-модель, який представляє собою отримані результати експерименту. Він містить у собі необхідні поля для подальшого запису у файл результатів.

Клас CSVWriter відповідає за запис результатів експерименту у визначений файл формату .CSV.

3.2.5 Використані принципи проектування

В ході проектування системи були використані принципи YAGNI та DRY.

Принцип YAGNI ("You Aren't Gonna Need It" або "Вам це не знадобиться") – це практика розробки програмного забезпечення, яка стверджує, що функції слід

додавати лише за потреби. Як частина філософії екстремального програмування (XP), YAGNI зменшує надлишки та неефективність у розробці, щоб сприяти бажаній збільшеній частоті випусків [44].

Принцип допомагає розробникам уникати марних зусиль щодо функцій, які, як вважають, знадобляться в певний момент. Ідея полягає в тому, що це припущення часто виявляється неправильним. Навіть якщо функція виявляється бажаною, все одно може виявитись, що реалізація не потрібна. Аргумент полягає в тому, щоб розробники не витрачали час на створення сторонніх елементів, які можуть бути не потрібні і можуть перешкоджати або сповільнювати процес розробки.

Оскільки YAGNI допомагає уникати витрат часу на функції, які можуть не використовуватися, основні функції програми краще розробляються і менше витрачається загального часу на кожний реліз.

Принцип «Don’t Repeat Yourself» (DRY) стверджує, що дублювання в логіці слід усувати абстракцією; дублювання в процесі слід усунути за допомогою автоматизації [45].

Додавання додаткового непотрібного коду до кодової бази збільшує обсяг роботи, необхідної для розширення та обслуговування програмного забезпечення в майбутньому. Дублікат коду збільшує технічний борг. Незалежно від того, чи дублювання випливає із програмування копіювальної вставки, або поганого розуміння того, як застосовувати абстракцію, це знижує якість коду. Дублювання в процесі також є марним результатом, якщо це можна автоматизувати. Тестування вручну, процеси ручного складання та інтеграції тощо повинні бути виключені, коли це можливо, за допомогою автоматизації.

Деякі поради щодо слідування цьому принципу [46]:

1. Завжди думайте про свій код як про серію менших модулів, які працюватимуть спільно між собою.

2. Подумайте про код, який буде багаторазовим, і спробуйте сформувати класи корисних програм.

3. Розділіть свою логіку та зафіксуйте їх у короткій послідовності коду.

4. Якщо ви хочете, щоб ваш код писався об’єктно-орієнтовано, подумайте про поведінку, яку можна класифікувати в класах.

5. Пам'ятайте – менше коду, менше ремонтопридатності.

3.3 Стратегія тестування

Тестування програмного забезпечення охоплює цілий ряд видів діяльності, дуже аналогічний послідовності процесів розробки програмного забезпечення. Сюди входять постановка задачі для тесту, проектування, написання тестів, тестування тестів і, нарешті, виконання тестів і вивчення результатів тестування. Вирішальну роль грає проектування тесту. Можливий цілий спектр підходів до вироблення стратегії проектування тестів [47].

Стратегія тестування повинна відповідати на такі питання:

- що тестувати;
- якими методами.

Усі методи тестування можна поділити на дві групи: тестування за вхідними даними – «чорного ящика» [48], та тестування логіки програми «білого ящика» [49]).

Тестування чорної скриньки може бути використано для методів, які мають лінійну структуру. Для того, щоб визначити, який метод тестування на чорну скриньку є більш ефективним у цій ситуації, спочатку слід розглянути кожен метод окремо.

Деякі методи з нелінійною структурою, особливо ті, що обробляють дані зі складною структурою, слід перевірити двома методами: еквівалентним методом розділу ("чорний ящик"), оскільки він дозволяє класифікувати можливі помилки, а іноді дозволяє виявляти такі помилки, які не були виявлені під час тестування методами «білого ящика», а також вхідні та вихідні дані не обмежуються специфікацією програми, а також залежать одна від одної та способу охоплення рішень.

Було вирішено, що програма буде тестуватися за допомогою методу припущення про помилку, який відноситься до методів «чорного ящика».

3.3.1 Тестування програми методом припущення про помилку

Оскільки метод припущення про помилку відноситься до методів «чорного ящика», то треба допустити, що ми не маємо доступу до вихідного коду.

Для проведення деякого експерименту користувач спочатку повинен ввести параметри експерименту. Він повинен заповнити такі поля, як ширина та висота зображення, кількість циклів запуску HSR-алгоритму. Інші елементи інтерфейсу передбачають, що буде обраний лише один з запропонованих варіантів (під час вибору алгоритму та файлу моделі), а значить користувач не зможе зробити тут щось непередбачуване.

Отже, можна припустити, що ввівши некоректну інформацію в вищезгаданих поля, можна зашкодити системі. На табл. 3.1 зображено результати тестування.

Таблиця 3.1 – Тестування методом припущення про помилку

Сценарій	Очікуваний результат	Актуальний результат
1	**2**	**3**
Поля «Ширина» та «Висота» залишилися не заповненими	Повідомлення про те, що потрібно заповнити усі поля	Повідомлення про те, що потрібно заповнити усі поля
Поле «Кількість тестів» містить негативне число	Повідомлення про те, що кількість тестів не може бути негативною	Повідомлення про те, що кількість тестів не може бути негативною
Поля «Ширина» або «Висота» містять негативне число або 0	Повідомлення про те, що розмір зображення повинен бути більше 0	Повідомлення про те, що розмір зображення повинен бути більше 0
Поле «Кількість тестів» містить 0 або пусте	Запускається рендеринг тривимірної моделі, але без тестування алгоритму	Запускається рендеринг тривимірної моделі, але без тестування алгоритму

Продовження таблиці 3.1

1	2	3
Поля «Ширина», «Висота» та «Кількість тестів» містять позитивне число	Запускається рендеринг тривимірної моделі з тестуванням алгоритму та записом результату у .csv-файл	Запускається рендеринг тривимірної моделі з тестуванням алгоритму та записом результату у .csv-файл

3.3.2 Налагодження програми

Оскільки система має багато функцій, зокрема HSR-алгоритмів, які містять безліч циклів, а система є досить об’ємною, налагодження програми було проведено за методом індукції та просування від місця виникнення помилки.

Для налагодження програми були використані можливості редактору «Visual Studio». Цей редактор дозволяє переглянути стек викликів (див. рис. 3.6) [50], за допомогою якого можна визначити, з якої саме точки було викликано деякий метод, в якому виникла помилка. Це дозволить зрозуміти, які вхідні дані було передано у цей метод і чому.

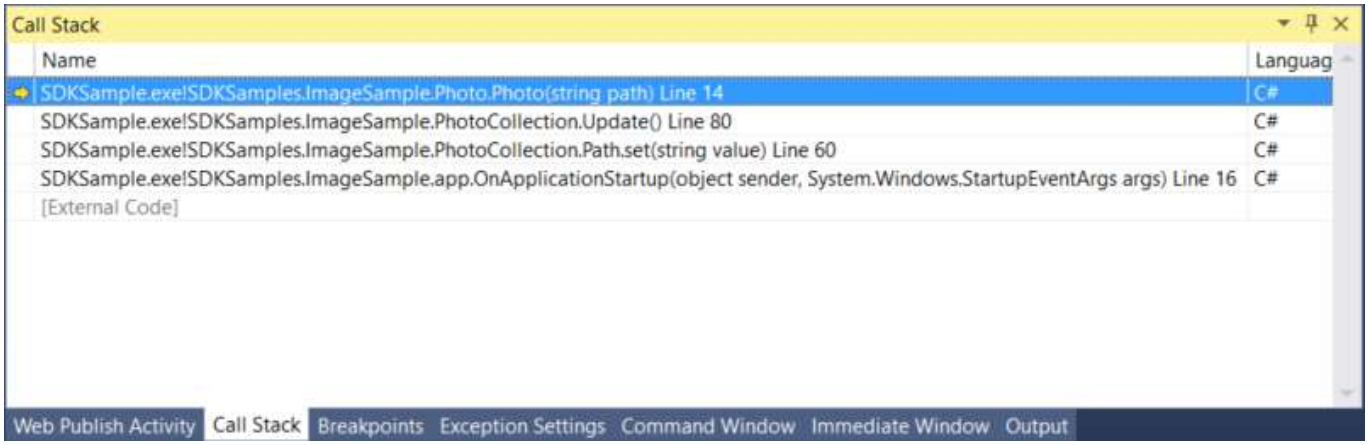

Рисунок 3.6 – Стек викликів

Також редактор має таку корисну функцію, як встановлення точок зупину («breakpoint») [51]. Краще за все ставити їх за декілька кроків до можливої помилки, як на рис. 3.7. Це дозволить краще зрозуміти, що посприяло появі помилки, а потім виправити її та протестувати цю ділянку коду ще раз. Для налагодження ділянок коду з великою кількістю циклів, є сенс використовувати умовні точки зупину [51]. Їх

використання дозволяє зупиняти роботу програми лише при деяких умовах, а не кожного разом. Це буде дуже корисно при налагодженні ділянок коду всередені циклів з 10 та більше ітерацій, щоб не зупинятися на кожній з них.

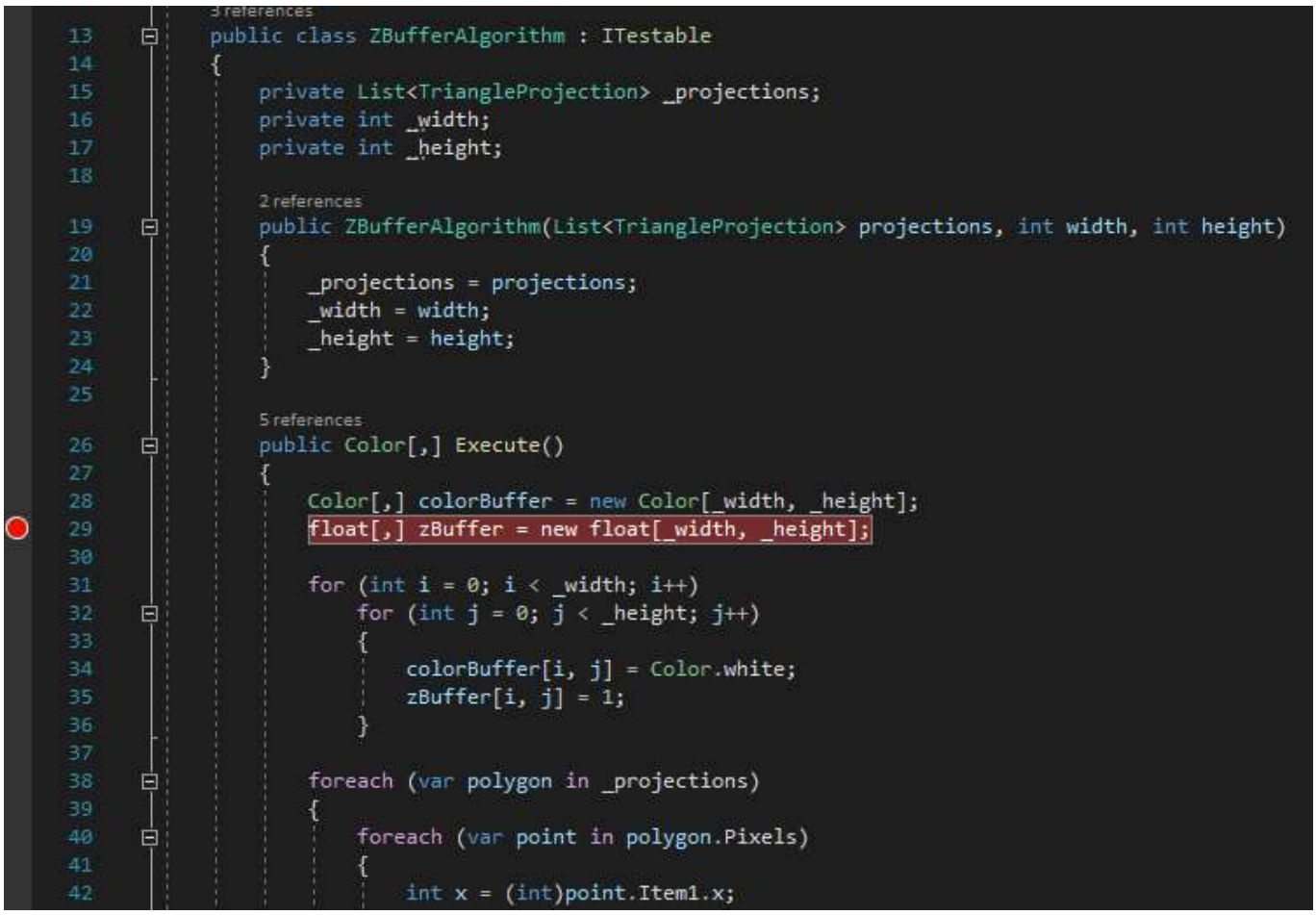

Рисунок 3.7 – Встановлення точки зупину

У випадку, коли потрібно переглянути результат загалом, особливо координати великої кількості точок, є сенс використовувати консоль для виводу даних. Якщо переглядати значення змінних вручну, це може зайняти купу часу. В таких ситуаціях було використано вбудовану функцію движка Unity – Debug.Log() [52]. Вона дозволяє вивести задану інформацію у вікно редактора, як зображено на рис. 3.8. Такий підхід дозволяє загалом переглянути результат та швидко виявити підозрілі результати, а потім за допомогою умовної точки зупину зупинитися на ділянці коду, де було знайдено підозрілий результат, та переглянути джерело помилки більш детально.

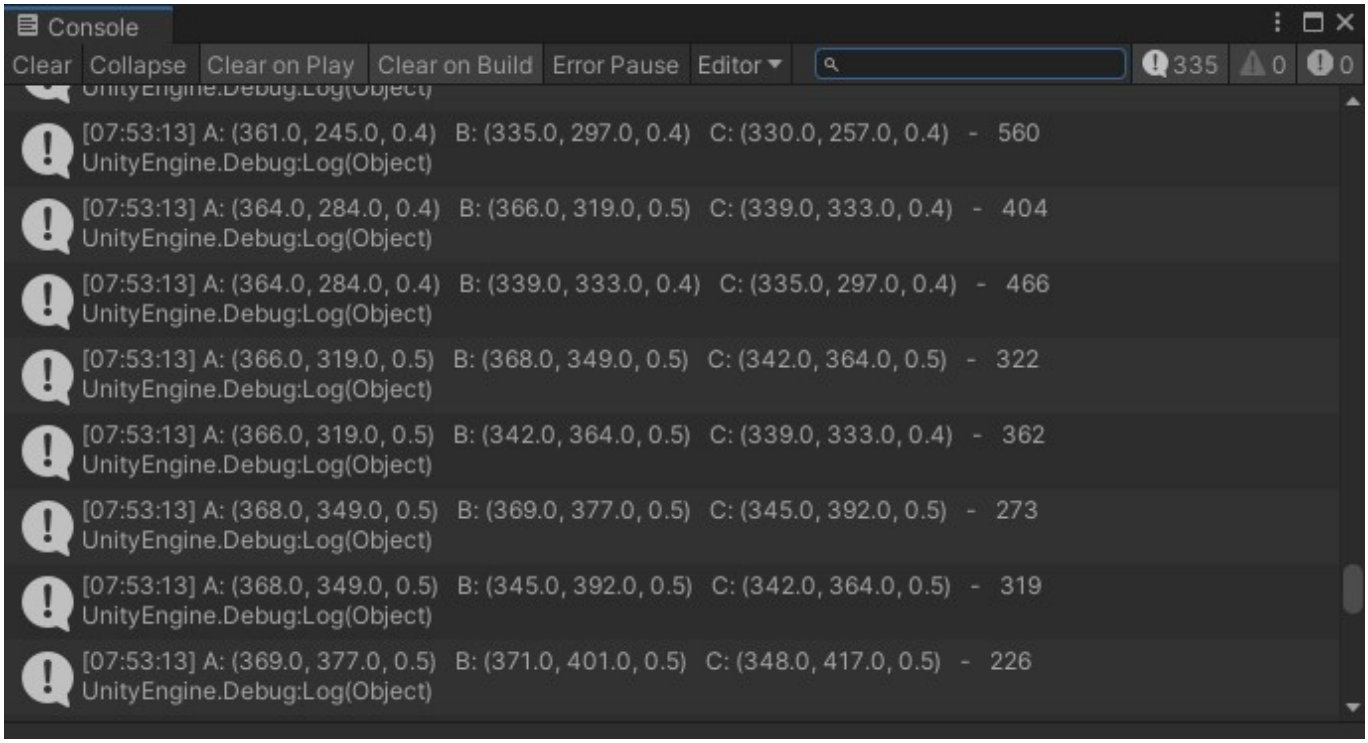

Рисунок 3.8 – Вивід результату у консоль

В ході налагодження програми було виявлено декілька помилок у логіці HSR-алгоритмів та рендеру тривимірної моделі. Всі виявлені помилки були виправлені.

Висновки до розділу 3

У третьому розділі було детально розглянуто аспекти зовнішнього та внутрішнього проектування.

Під час зовнішнього проектування було формалізовано задачу, визначені вхідні та вихідні параметри та розроблена схема способів використання. Також було розглянуто систему у динаміці за допомогою розробленої діаграми послідовності. Крім цього був розроблений простий інтерфейс користувача після визначення основних положень щодо розробки інтуїтивно зрозумілого користувацького інтерфейсу.

При внутрішньому проектуванні було обрано технологічну платформу, мову програмування та систему контролю версій. Було розроблено ієрархію класів та їх взаємодію згідно з принципами парадигми ООП.

Наостанок були розглянуті методи тестування ПО. Розроблена програма була протестована одним з цих методів, а також було проведено її налагодження.

Програма є придатною до використання.

4 ДОСЛІДЖЕННЯ ЧАСОВОЇ ЕФЕКТИВНОСТІ ОБРАНИХ АЛГОРИТМІВ ВИДАЛЕННЯ НЕВИДИМИХ ЛІНІЙ ТА ПОВЕРХОНЬ

4.1 Підготовка до експерименту

4.1.1 Опис використаного програмно-апаратного середовища

Дослідження проводилося з використанням персонального комп’ютеру на базі операційної системи Windows. Нижче описані деякі важливі характеристики вищезгаданого ПК.

Процесор:

- модель: Intel Core i5-8250U;
- кількість ядер: 4;
- тактова частота процесора: від 1.6 до 3.4 ГГц;
- об'єм кеш-пам'яті: 6 МБ.

Оперативна пам'ять:

- модель: HyperX SODIMM DDR3L-1600 8192MB PC3-12800;
- тип пам’яті: SODIMM DDR3L;
- частота пам’яті: 1600 МГц;
- пропускна здатність: 12800 МБ/с;
- ємність одного модуля оперативної пам'яті: 8 ГБ;
- кількість модулів оперативної пам'яті: 2;
- загальна ємність оперативної пам'яті: 16 ГБ.

Відеокарта:

- модель: NVIDIA GeForce GTX 1050;
- тип пам’яті: GDDR5;
- інтерфейс: PCI Express 3.0;
- обсяг пам’яті: 2 ГБ;
- частота роботи GPU: 1455 МГц;
- частота роботи пам’яті: 1752 МГц.

Пам’ять диску:

- модель: Kingston A2000 250GB;
- об’єм пам’яті: 250 ГБ;
- інтерфейс підключення: PCI Express 3.0 x4;
- швидкість читання: до 2000 МБ/с;
- швидкість запису: до 1100 МБ/с;
- тип пам’яті: GDDR5.

4.1.2 Опис підходу для визначення часу роботи алгоритму

Одразу слід зазначити, що у експериментах буде визначений час роботи саме алгоритму, а не всього циклу рендерингу тривимірної моделі. Для заміру часу алгоритмів було розроблено спеціальний статичний клас з методом, який може запустити метод Execute будь-якого об'єкту класу, який реалізує інтерфейс ITestable.

Визначення часу проводилося за допомогою вбудованих можливостей мови C#. А саме було використано клас Stopwatch, який має безліч можливостей та може заміряти час з точністю до мілісекунди [53].

Також було реалізовано можливість визначати час з більшою точністю, якщо час роботи алгоритму є меншим за одну мілісекунду. Це робиться таким чином: підраховується час роботи алгоритму визначену кількість разів, а потім цей час поділяється на цю саму кількість разів, після чого отримуємо остаточний результат. Але як виявилося на практиці, найчастіше час роботи алгоритму значно перевищує одну мілісекунду, тому можна визначати час роботи кожного випадку окремо.

Після того, як система визначає час роботи алгоритму, вся отримана інформація про експеримент та час роботи записується в окремий файл формату .CSV з результатами у вигляді, як зображено на рис. 4.1. Потім дослідник може відкрити цей файл та обробляти отриману інформацію самостійно.

Algorithm	Width	Height	Polygons count	Pixels count	Result (ms)
ZBuffer	800	800	11880	635648	82
ZBuffer	800	800	11880	635648	87
ZBuffer	800	800	11880	635648	104
ZBuffer	800	800	11880	635648	85
ZBuffer	800	800	11880	635648	82
ZBuffer	800	800	11880	635648	82
ZBuffer	800	800	11880	635648	93
ZBuffer	800	800	11880	635648	98
ZBuffer	800	800	11880	635648	84
ZBuffer	800	800	11880	635648	87
ZBuffer	800	800	11880	635648	83

Рис. 4.1 – Приклад вмісту файлу з результатами

4.1.3 Вплив збірки сміття на результати вимірювань

Під час проведення експерименту дуже важливо, щоб зовнішні фактори якомога менше впливали на отримані результати. В C# є така корисна функція, як збірка сміття. Вона корисна тим, що коли система потребує виділення пам’яті на щось, середа CLR автоматично запускає збірку сміття, яка видаляє з купи ті ділянки пам’яті, що вже не використовуються [54].

Але під час проведення експерименту це може негативно вплинути на результати, бо розробник зазвичай не контролює конкретний момент, коли збірка сміття буде запущена. Може статися така ситуація, коли під час вимірювання часу роботи алгоритму спрацьовує збірка мусора та впливає на результат. Результати такого випадку представлено на рис. 4.2.

ScanLine	1000	1000	8	879332	37
ScanLine	1000	1000	8	879332	102
ScanLine	1000	1000	8	879332	37
ScanLine	1000	1000	8	879332	37
ScanLine	1000	1000	8	879332	37
ScanLine	1000	1000	8	879332	37
ScanLine	1000	1000	8	879332	37
ScanLine	1000	1000	8	879332	37
ScanLine	1000	1000	8	879332	100
ScanLine	1000	1000	8	879332	37
ScanLine	1000	1000	8	879332	37
ScanLine	1000	1000	8	879332	37
ScanLine	1000	1000	8	879332	37
ScanLine	1000	1000	8	879332	37

Рисунок 4.2 – Результат впливу збірки сміття на вимірювання часу

На малюнку видно, що час виконання алгоритму (остання колонка) є стабільним і майже завжди дорівнює 37 мс. Але іноді час виконання зростає у декілька разів, що виглядає дуже підозріло. Крім того, зріст часу виконання відбувається чітко кожні N ітерацій, що наводить на думку, що це не збіг.

Як виявилося в конкретно цьому випадку на початку роботи алгоритму створювався буфер кольору великих розмірів. Після декількох ітерацій в купі не залишалося вільних ділянок пам’яті для створення чергового буферу, тому CLR запускала збірку сміття прямо під час роботи алгоритму, що дуже вплинуло на час виконання.

Тому, щоб запобігти таких ситуацій слід вручну запускати збірку сміття за межами роботи таймеру для заміряння часу роботи алгоритму. Це можна зробити за допомогою вбудованого методу GC.Collect() [55].

4.2 Проведення експерименту

Весь експеримент можна поділити на три етапи:

- визначення залежності від кількості полігонів;
- визначення залежності від кількості пікселів для обробки;
- визначення залежності від роздільної здатності екрану.

На першому етапі експерименту були використані моделі, які описані у пункті 2.5. Візуальне представлення цієї моделі зображено на рис 2.1. Кожному HSR-алгоритму на вхід подавали модель кулі різної деталізації та замірювали час роботи. Було підібрано 12 версій кулі з різною деталізацією від 80 до 39 600 полігонів. Результати даного етапу експерименту представлено в табл. 4.1.

Таблиця 4.1 – Результати етапу експерименту визначення залежності від кількості полігонів

Висота	Ширина	Кількість полігонів	Кількість пікселів до обробки	Час роботи алгоритму Z-буфера (мс)	Час роботи scan-line алгоритму (мс)	Час роботи алгоритму «художника» (мс)
1	**2**	**3**	**4**	**5**	**6**	**7**
800	800	80	309 265	8.14	6.04	5.22

Продовження таблиці 4.1

1	2	3	4	5	6	7
800	800	1 890	396 776	21.1	9.74	20.68
800	800	4 224	442 422	26.56	11.58	30.8
800	800	6 240	479 329	25.48	12.62	44.58
800	800	8 740	521 360	29.5	13.66	60.46
800	800	11 880	572 212	25.44	14.2	82.08
800	800	15 250	624 873	28.5	15.32	108.6
800	800	19 320	685 437	32.42	17.06	139.46
800	800	23 560	746 257	36.24	15.9	172.72
800	800	28 560	817 276	41	17.76	209.54
800	800	33 670	889 481	47.08	18.26	248.92
800	800	39 600	970 228	50.6	18.32	291.22

Під час проведення другого етапу експерименту було використано моделі, зображення яких представлено на рис. 2.2. На цьому етапі в якості вхідних даних була використана малополігональна модель площини різних розмірів. В залежності від розміру моделі змінюється й кількість пікселів, які потрібно обробити алгоритму видалення невидимих ліній та поверхонь, що і потрібно для визначення на цьому етапі. Результати даного етапу експерименту представлено у табл. 4.2.

Таблиця 4.2 – Результати етапу експерименту визначення залежності від кількості пікселів, які потрібно обробити

Висота	Ширина	Кількість полігонів	Кількість пікселів до обробки	Час роботи алгоритму Z-буфера (мс)	Час роботи scan-line алгоритму (мс)	Час роботи алгоритму «художника» (мс)
1	**2**	**3**	**4**	**5**	**6**	**7**
800	800	8	22 840	3.16	5.31	1.47
800	800	8	63 000	4.06	5.3	1.94
800	800	8	123 168	5.85	6.48	2.73

Продовження таблиці 4.2

1	**2**	**3**	**4**	**5**	**6**	**7**
800	800	8	203 336	9.26	10.44	5.46
800	800	8	303 496	11.43	10.43	5.93
800	800	8	423 298	14.37	15.17	7.89
800	800	8	563 388	18.51	17.19	9.86

Для проведення останнього етапу експерименту були використана тривимірна модель чайника, яка зображена на рис 2.3. На цьому етапі було використано одну й ту саму модель чайника, змінювалася лише ширина та висота полотна, на якому потрібно було відобразити обрану модель. В табл. 4.3 представлено результати останнього етапу експерименту.

Таблиця 4.3 – Результати етапу експерименту визначення залежності від роздільної здатності екрану

Висота	Ширина	Кількість полігонів	Кількість пікселів до обробки	Час роботи алгоритму Z-буфера (мс)	Час роботи scan-line алгоритму (мс)	Час роботи алгоритму «художника» (мс)
200	200	10 712	144 693	8.94	1.01	62.71
400	400	10 712	231 145	12.02	4.04	65.38
600	600	10 712	339 655	22.1	7.6	69.02
800	800	10 712	474 413	21.74	13.12	73.62
1000	1000	10 712	639 011	28.04	19.32	77.1
1200	1200	10 712	833 039	35.24	24.1	82.74
1400	1400	10 712	1 059 555	44.38	31.9	87.48

4.3 Результати експерименту

В ході проведення експерименту було помічено, що незважаючи на те, що на кожному етапі проводилися тести для визначення залежності алгоритму від конкретної характеристики, при зміні інших показників завжди змінюється показник «Кількість пікселів до обробки».

Наприклад, в експерименті з кулею різної деталізації з ростом кількості полігонів моделі зростала і кількість пікселів, які потрібно обробити. І хоча візуально більш деталізована модель займала стільки ж місця, скільки і менш деталізована, кількість пікселів все одно зростала. Це відбувається з тієї причини, що двічі враховуються пікселі, які знаходяться на ребрах. Тобто, зі зростом кількості полігонів, зростала й кількість ребер, де пікселі враховувалися двічі, а значить зростала і загальна кількість пікселів. Експеримент було проведено таким чином, щоб якомога більше виключити цей фактор, але він все одно вплинув на результати.

Інший приклад, зі зростанням роздільної здатності екрану. Тут все набагато простіше – чим більше здатність екрану, тим більше пікселів відображається на екрані, хоча модель для тестування не змінюється. В цьому експерименті роздільна здатність екрану більше вплинула на зріст кількості пікселів для обробки, аніж в експерименті з визначенням залежності від кількості полігонів.

І лише в експерименті з визначенням залежності від кількості пікселів до обробки ніякі показники не змінюються, окрім показника, який досліджується. Тому слід починати аналіз результатів саме цього експерименту, щоб результати аналізу можна було використати при аналізі інших двох.

4.3.1 Аналіз впливу зміни кількості пікселів до обробки на час роботи алгоритму

За допомогою засобів Excel на основі отриманих даних був побудований графік, який візуально відображає залежність часу роботи від кількості пікселів для обробки для кожного алгоритму. Також були побудовані лінії тренду до кожної лінії, а також були визначені функції цих ліній тренду. Це дозволить зрозуміти, наскільки змінюється час алгоритму від того чи іншого показника. Побудований графік з лініями тренду зображено на рис. 4.3.

Як видно з нижче приведеного малюнку для всіх алгоритмів залежність часу від кількості пікселів до обробки є лінійною. Це можна підтвердити високим коефіцієнтом детермінації R^2, який для усіх ліній тренду є вищим за 0.95.

Визначені функції для кожної лінії тренду показують на те, як саме змінюється час роботи алгоритму в залежності від кількості пікселів.

Найменшу залежність від цього показника демонструє алгоритм «художника». В середньому час роботи цього алгоритму підвищується на 1.58 мс з кожними 100 тисяч пікселів.

Наступним по порядку йде scan-line алгоритм, час роботи якого підвищується на 2.35 мс за кожні 100 тисяч нових пікселів.

І найбільш залежним від цього показника виявився алгоритм Z-буфера, який має результат в 2.85 мс за кожні 100 тисяч пікселів.

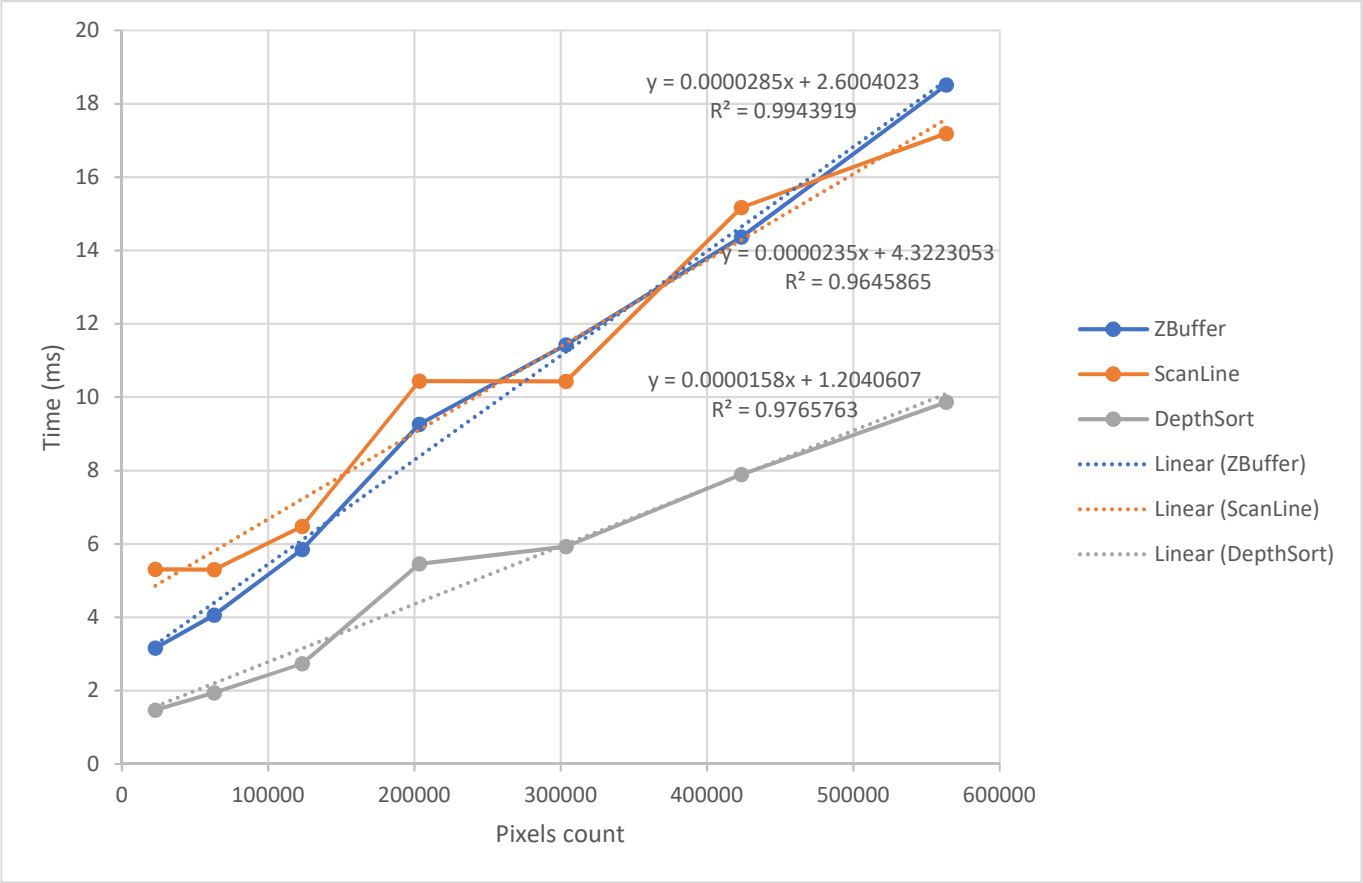

Рисунок 4.3 – Зміна часу роботи в залежності від кількості пікселів

Але слід зауважити, що HSR-алгоритми демонструють такі результати саме на персональному комп'ютері, на якому проводиться дослідження. Якщо провести експеримент на іншій ЕОМ, вищезазначені коефіцієнти будуть пропорційно змінюватися відносно один одного. Тому є сенс виявити, на який відсоток один алгоритм краще або гірше іншого в деякому контексті. Результати такого підрахунку зображені в табл. 4.4. Результати приведені у таблиці слід трактувати таким чином: коефіцієнт впливу алгоритму «художника» дорівнює 55% від коефіцієнту алгоритму Z-буфера.

Таблиця 4.4 – Величина коефіцієнту впливу відносно інших алгоритмів

Алгоритм	Z-буфера	Scan-line	«Художника»
Z-буфера	100%	121%	180%
Scan-line	82%	100%	149%
«Художника»	55%	67%	100%

4.3.2 Аналіз впливу зміни кількості полігонів на час роботи алгоритму

Як і для результатів попереднього експерименту, був побудований графік, який демонструє вплив зміни кількості полігонів на час роботи алгоритму. Але тут слід зауважити, що на час впливає ще й показник «Кількість пікселів до обробки», який було розглянуто у попередньому пункті. Побудований графік з лініями тренду зображено на рис. 4.4.

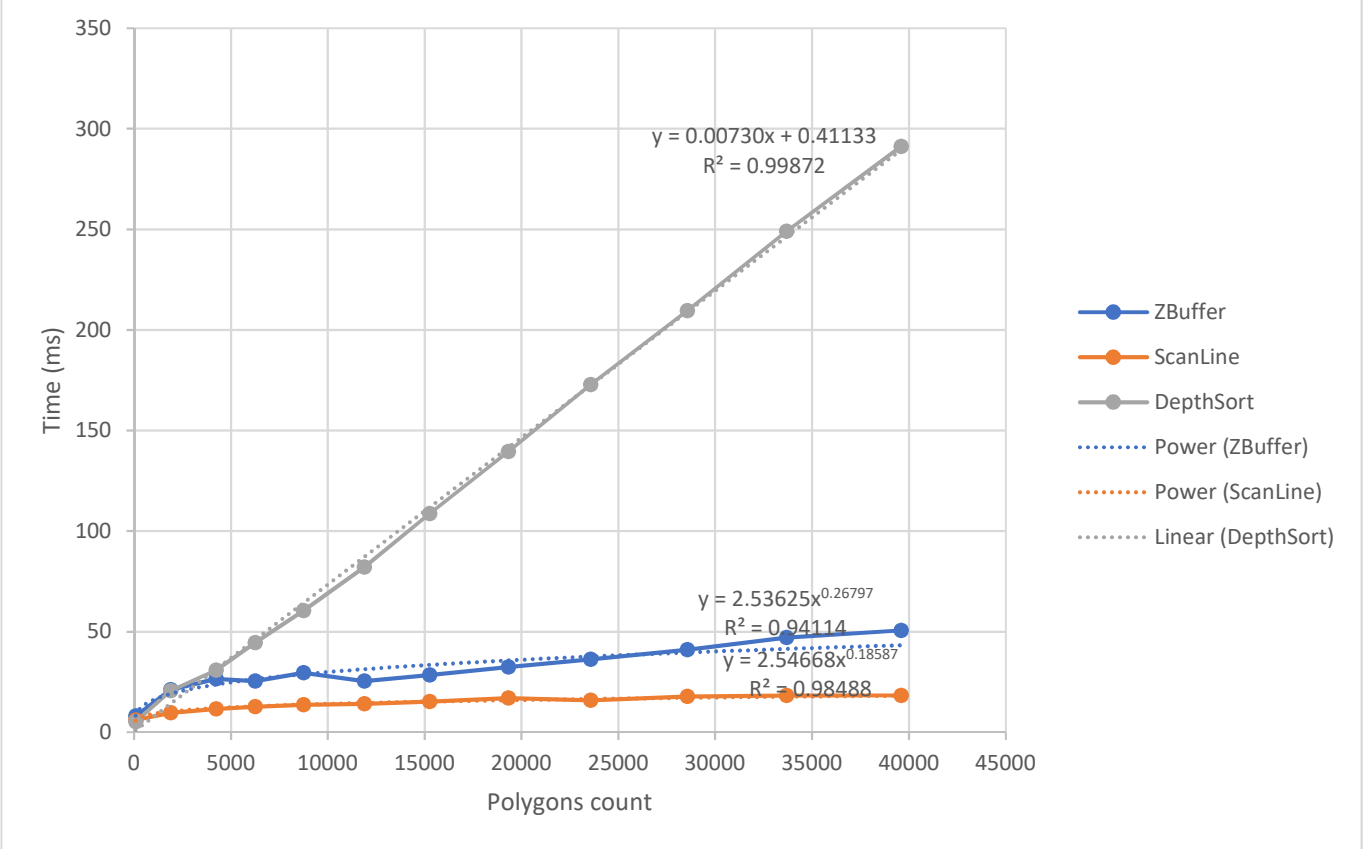

Рисунок 4.4 – Зміна часу роботи в залежності від кількості полігонів

З графіку добре видно, що алгоритм «художника» є дуже чутливим до кількості полігонів. Але робити якісь висновки щодо інших двох алгоритмів доволі важко, тому що їх час роботи змінюється не так різко, враховуючи, що на час також міг вплинути й зріст кількості пікселів до обробки більше ніж в 3 рази. Якщо відняти час, який зріс

саме через підвищення кількості, то можна отримати зовсім інші результати. Тому було побудовано ще один графік з врахуванням цього факту. На рис. 4.5 зображено графік залежності часу роботи алгоритмів лише від кількості полігонів.

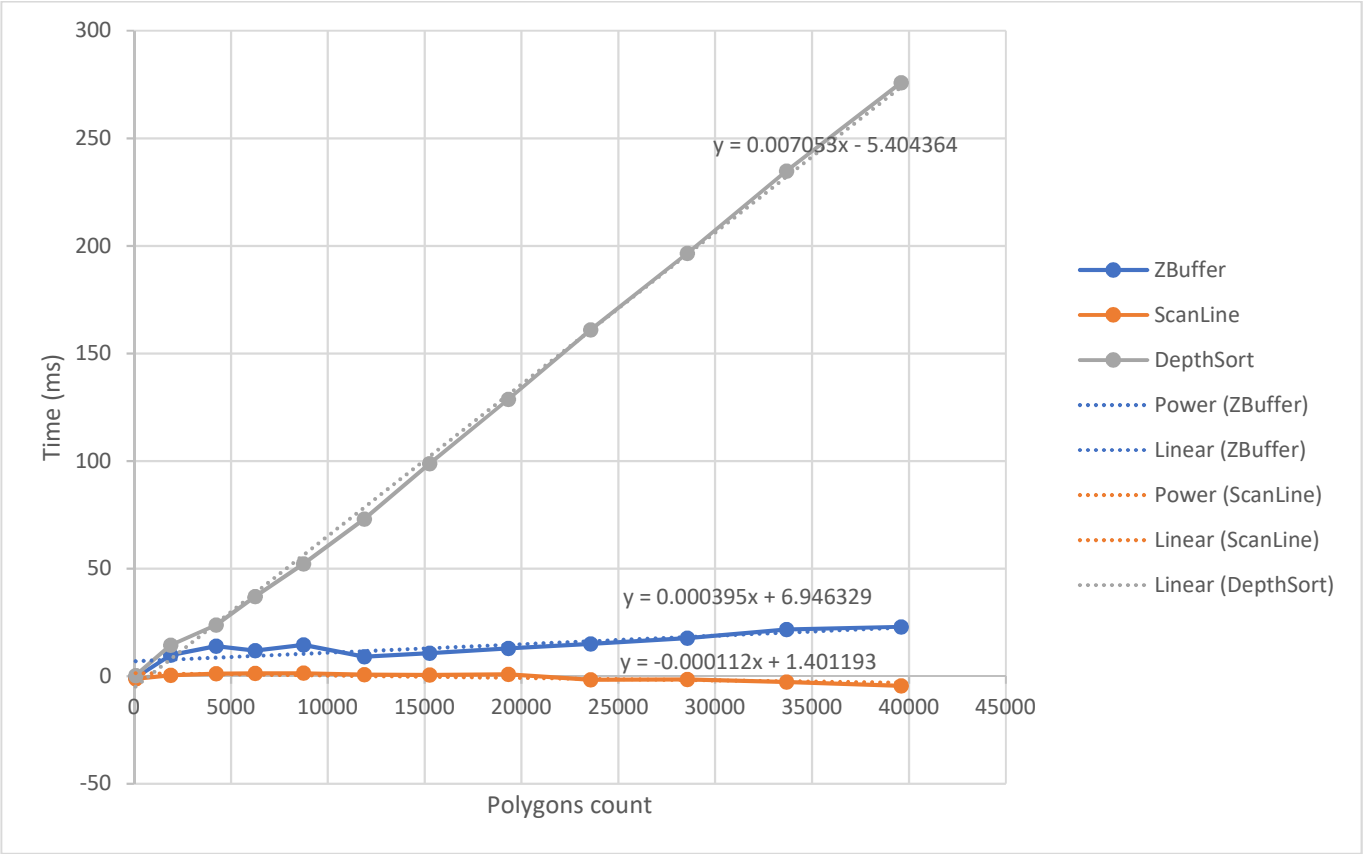

Рисунок 4.5 – Зміна часу роботи в залежності лише від кількості полігонів

З нового графіку видно, що час алгоритму «художника» досі дуже залежить від кількості полігонів. А от щодо інших двох ситуація змінилася. Лінія тренду scan-line алгоритму направлена вниз, що свідчить про те, що цей алгоритм взагалі не залежить від кількості полігонів. Лінія тренду має бути рівною, а коефіцієнт зміни дорівнювати нулю, а не від'ємному числу. Скоріше за все так сталося через деяку похибку при вимірюванні, бо коефіцієнт впливу наближений до нуля. Що стосується алгоритму Z-буфера, то він все ж таки залежить від зміни кількості полігонів, хоча й зовсім трохи.

Загалом ми маємо такі коефіцієнти впливу:

- час роботи алгоритму «художника» збільшується на 70.53 мс з кожними 10 тисяч полігонів;
- час роботи scan-line алгоритму не змінюється взагалі, тобто коефіцієнт зміни дорівнює 0;

– час роботи алгоритму Z-буферу збільшується на 3.95 мс з кожними 10 тисяч полігонів.

В табл. 4.5 наведено ефективність HSR-алгоритмів відносно одне одного.

Таблиця 4.5 – Величина коефіцієнту впливу відносно інших алгоритмів

Алгоритм	Z-буфера	Scan-line	«Художника»
Z-буфера	100%	-	6%
Scan-line	-	-	-
«Художника»	1786%	-	100%

4.3.3 Аналіз впливу зміни роздільної здатності екрану на час роботи алгоритму

Як було зазначено раніше, при зростанні роздільної здатності екрану зростає й кількість пікселів до обробки, тому потрібно брати до уваги цей факт при аналізі. На рис. 4.6 зображено побудований графік залежності часу роботи алгоритмів від роздільної здатності екрану без урахування цього факту.

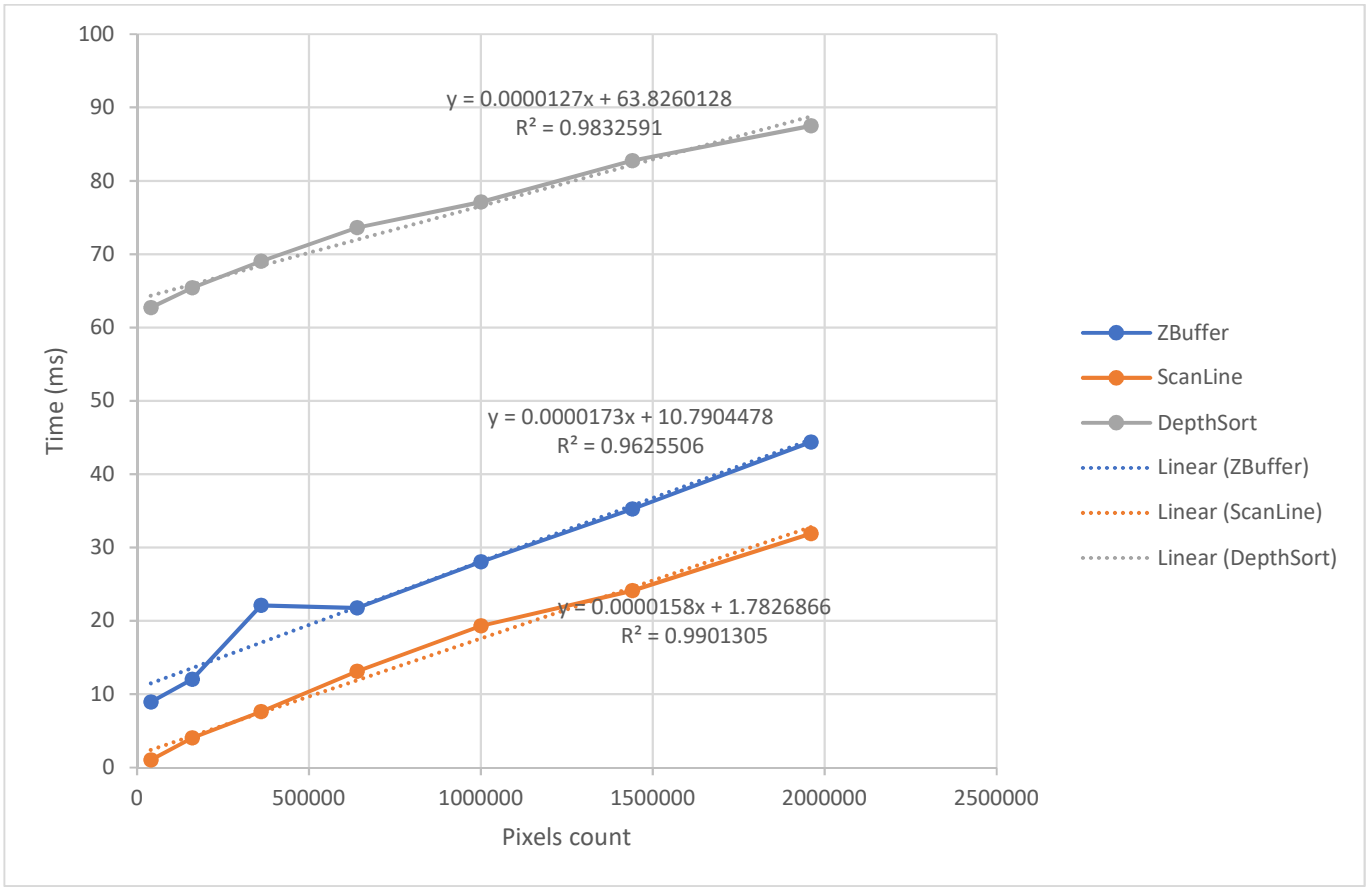

Рисунок 4.6 – Зміна часу роботи в залежності від роздільної здатності екрану

З цього графіку видно, що час роботи усіх алгоритмів більш-менш однаково зростає з роздільною здатністю екрану. Але точно сказати не можна, бо треба позбавитися від впливу кількості пікселів до обробки, яка невпинно збільшувалася з роздільною здатністю екрану. Тому було побудовано ще один графік, який зображено на рис. 4.7, який відображає залежність часу роботи лише від роздільної здатності екрану.

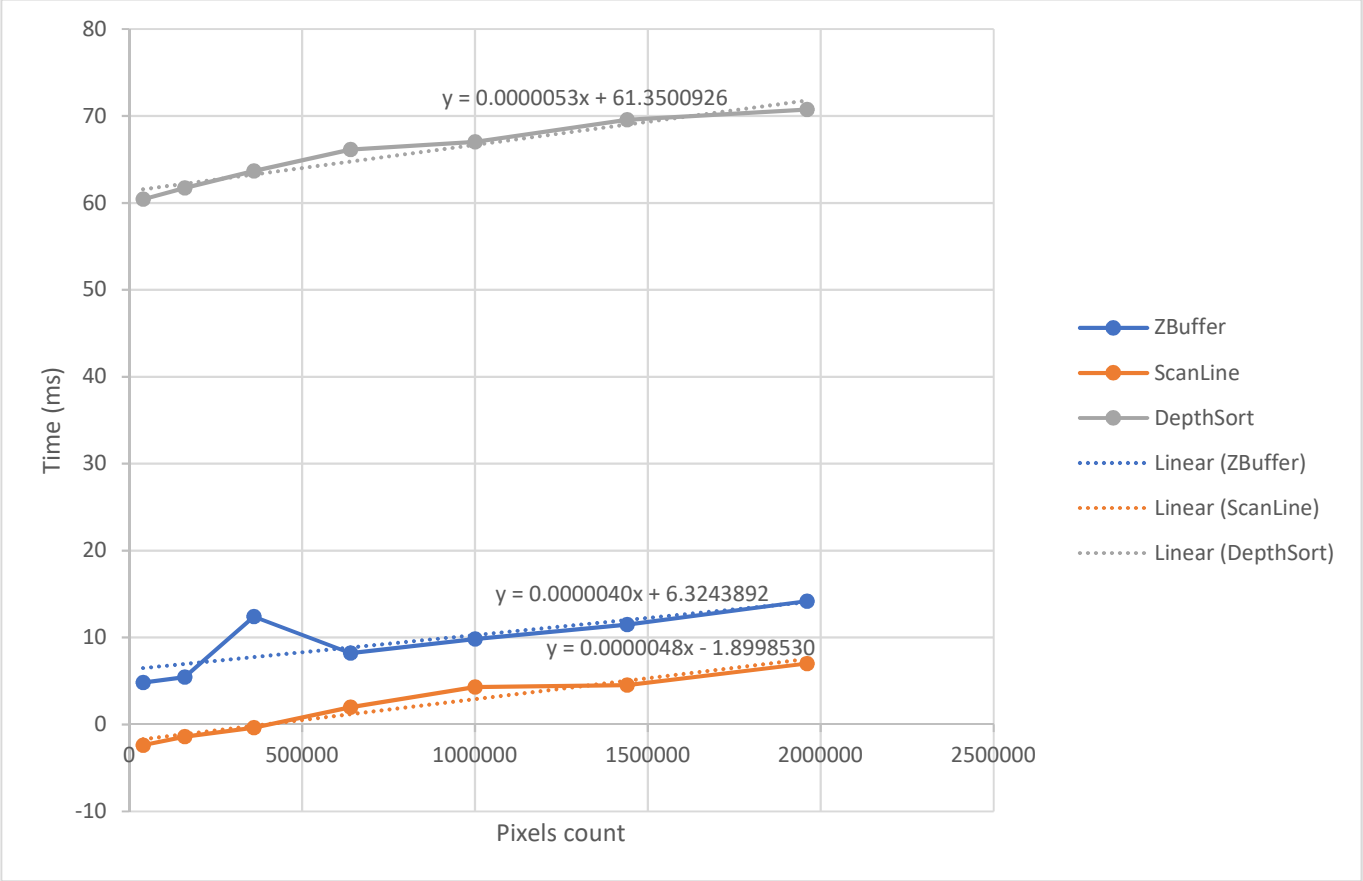

Рисунок 4.7 – Зміна часу роботи в залежності лише від роздільної здатності екрану

З графіку видно, що кожен з алгоритмів зовсім трохи залежить від роздільної здатності екрану. Менше за всіх від цього показника залежить алгоритм Z-буфера, а більше – алгоритм «художника». Але різниця між ними настільки мала, що майже не грає ролі.

Загалом ми маємо такі коефіцієнти впливу:

– час роботи алгоритму «художника» збільшується на 5.3 мс з кожним мільйоном пікселів;

– час роботи scan-line алгоритму збільшується на 4.8 мс з кожним мільйоном пікселів;

– час роботи алгоритму Z-буферу збільшується на 4 мс з кожним мільйоном пікселів.

В табл. 4.6 наведено ефективність HSR-алгоритмів відносно одне одного.

Таблиця 4.6 – Величина коефіцієнту впливу відносно інших алгоритмів

Алгоритм	Z-буфера	Scan-line	«Художника»
Z-буфера	100%	83%	75%
Scan-line	120%	100%	91%
«Художника»	133%	110%	100%

4.3.4 Прогнозування часу роботи алгоритму на основі знайдених коефіцієнтів впливу

В пунктах 4.3.1-4.3.3 було визначено, як саме роздільна здатність екрану, кількість пікселів до обробки та кількість полігонів впливають на час роботи алгоритмів видалення невидимих ліній та поверхонь. Для кожного показника було знайдено коефіцієнт впливу на час роботи алгоритму. Підсумкові результати з коефіцієнтами зображені у табл. 4.7.

Таблиця 4.7 – Підсумкова таблиця з коефіцієнтами впливу

Алгоритм	Коеф. впливу кількості пікселів до обробки	Коеф. впливу кількості полігонів	Коеф. впливу роздільної здатності екрану
Z-буфера	0.0000285	0.000395	0.000004
Scan-line	0.0000235	0	0.0000048
«Художника»	0.0000158	0.007053	0.0000053

Тепер маючи ці коефіцієнти можна обчислити приблизний час роботи алгоритму в залежності від вхідних даних за формулою 4.1:

$$T_e = K_{pix} * Pix + K_{pol} * Pol + K_{Res} * H * W, \quad (4.1)$$

де T_e – приблизний час роботи алгоритму,

K_{pix} – коефіцієнт впливу кількості пікселів до обробки на час роботи,

K_{pol} – коефіцієнт впливу кількості полігонів на час роботи,

K_{res} – коефіцієнт впливу кількості роздільної здатності екрану на час роботи,

Pix – кількість пікселів до обробки,

Pol – кількість полігонів,

H – висота зображення,

W – ширина зображення.

А тепер спробуємо обчислити приблизний час за формулою та зрівняти результат обчислення з фактичним результатом часу роботи алгоритму. В табл. 4.8 приведені деякі з цих порівнянь.

Таблиця 4.8 – Порівняння фактичних та обчислених результатів

Алгоритм	Висота	Ширина	Кількість полігонів	Кількість пікселів до обробки	Фактичний час роботи алгоритму (мс)	Обчислений час роботи алгоритму (мс)
1	**2**	**3**	**4**	**5**	**6**	**7**
Z-буферу	200	200	10 712	144 693	8.94	8.51
Z-буферу	400	400	10 712	231 145	12.02	11.46
Z-буферу	600	600	10 712	339 655	22.1	15.35
Z-буферу	800	800	10 712	474 413	21.74	20.31
Z-буферу	1000	1000	10 712	639 011	28.04	26.44
Z-буферу	1200	1200	10 712	833 039	35.24	33.73
Z-буферу	1400	1400	10 712	1 059 555	44.38	42.27
Scan-line	200	200	10 712	144 693	1.01	3.59
Scan-line	400	400	10 712	231 145	4.04	6.20
Scan-line	600	600	10 712	339 655	7.6	9.71
Scan-line	800	800	10 712	474 413	13.12	14.22
Scan-line	1000	1000	10 712	639 011	19.32	19.82
Scan-line	1200	1200	10 712	833 039	24.1	26.49
Scan-line	1400	1400	10 712	1 059 555	31.9	34.31
Художника	200	200	10 712	144 693	62.71	78.05
Художника	400	400	10 712	231 145	65.38	80.05

Продовження таблиці 4.8

1	2	3	4	5	6	7
Художника	600	600	10 712	339 655	69.02	82.83
Художника	800	800	10 712	474 413	73.62	86.44
Художника	1000	1000	10 712	639 011	77.1	90.95
Художника	1200	1200	10 712	833 039	82.74	96.35
Художника	1400	1400	10 712	1 059 555	87.48	102.68

З таблиці видно, що обчислений час роботи алгоритму дещо відрізняється від фактичного результату. Також слід мати на увазі, що отриманий фактичний результат міг зафіксуватися з деяким відхиленням через зовнішні фактори. Але в цілому обчислений час роботи більш-менш співпадає з фактичними результатами з приблизним відхиленням у 15-20%.

Висновки до розділу 4

У четвертому розділі було проведено декілька експериментів для визначення впливу кожного з трьох основних показників на час роботи алгоритмів видалення невидимих ліній та поверхонь. Під час проведення експерименту були зібрані всі необхідні дані.

Після аналізу зібраної інформації було з’ясовано, що на час роботи кожного з HSR-алгоритмів одні й ті самі показники впливають по-різному. Наприклад, кількість полігонів дуже впливає на час роботи алгоритму «художника» та зовсім не впливає на час роботи scan-line алгоритму. Щодо роздільної здатності екрану – цей показник впливає на час роботи усіх досліджуваних HSR-алгоритмів приблизно однаково. Показник «кількість пікселів до обробки» найменше впливає на час роботи алгоритму «художника» і майже в два рази більше – на алгоритм Z-буферу.

На основі отриманих даних було розраховано коефіцієнти впливу (див. табл. 4.7) для кожного показника, які чисельно відображають ступінь впливу на час роботи кожного алгоритму. Знайдені коефіцієнти впливу можна застосувати при реалізації методу-перемикача, який буде обирати найефективніший HSR-алгоритм в залежності від вхідних даних. Він буде обчислювати очікуваний час кожного з алгоритмів, а потім обирати той, що отримав найкращий результат.

5 ОХОРОНА ПРАЦІ ТА БЕЗПЕКА В НАДЗВИЧАЙНИХ СИТУАЦІЯХ

5.1 Вимоги безпеки при виконанні робіт на робочому місці

Для того, щоб робітник міг працювати на належному рівні, є необхідним його дотримування до певних правил роботи. Крім цього потрібно, щоб робоче середовище також відповідало визначеним нормам безпеки. В даний момент ці питання регулюються великою кількістю нормативно-правових актів з охорони праці (далі скорочено НПАОП) та державними санітарними правилами і нормами роботи (далі скорочено ДСанПіН).

Недотримування до вимог ДСанПіН та НПАОП може привести до підвищення ризику впливу шкідливих виробничих факторів на працівника. Тривалий вплив цих шкідливих факторів може негативно вплинути на ефективність праці, або навіть на здоров’я людини.

Оскільки під час розробки деякого програмного забезпечення програміст-розробник так чи інакше працює з комп’ютерною технікою, насамперед потрібно розглянути саме ті ДСанПіН та НПАОП, які стосуються роботи з відповідною технікою та засобами. Нижче наведені такі документи:

- НПАОП 0.700-7.15-18 Вимоги щодо безпеки та захисту здоров’я працівників під час роботи з екранними пристроями, затверджені від 14.02.2018 р. № 207 (далі скорочено НПАОП 0.700-7.15-18) [56];
- НПАОП 80.0-1.12-04 Правила безпеки під час навчання в кабінетах інформатики навчальних закладів системи загальної середньої освіти, затверджені від 20.11.2007 р. № 252 (далі скорочено НПАОП 80.0-1.12-04) [57];
- ДСанПіН 3.3.2-007-98 Державні санітарні правила і норми роботи з візуальними дисплейними терміналами електронно-обчислювальних машин, затверджені від 10.12.1998 р. № 7 (далі скорочено ДСанПіН 3.3.2-007-98) [58];
- Інструкція щодо надання послуг з ремонту побутової радіоелектронної апаратури, затверджена від 27.08.2000 р. № 20 [59].

Вимоги з НПАОП 0.700-7.15-18 загалом поділяються на три категорії:

- вимоги безпеки до робочих місць;

- вимоги безпеки під час роботи з екранними пристроями;
- вимоги безпеки до екранних пристроїв.

Вимоги безпеки до робочих місць:

- робочі місця повинні бути спроектовані таким чином, щоб у працівника був простір для рухів та зміни положення;
- випромінювання екранних пристроїв має бути зведено до допустимого рівня;
- освітлення робочого місця повинно створювати необхідний контраст між навколишнім середовищем та екраном та відповідати вимогам ДСанПіН 3.3.2-007-98 [58];
- мікроклімат приміщень з робочими місцями повинен підтримуватися на постійному рівні та відповідати вимогам ДСН 3.3.6.042-99 Санітарних норм мікроклімату виробничих приміщень [60];
- поверхня робочого столу повинна бути з низькою відбивною здатністю, а робочий стіл – достатнього розміру.

Вимоги безпеки під час роботи з екранними пристроями:

- кожний день потрібно прибирати пристрій від пилу або інших забруднень;
- електроні пристрої слід вимикати з електричної мережі в кінці робочого дня;
- також їх слід вимикати у випадку аварійної ситуації;
- не можна виконувати ремонт екранних пристроїв під час роботи на робочому місці:
- не можна працювати з екранними пристроями, які ведуть себе не передбачувано.

Вимоги безпеки до екранних пристроїв:

- екранні пристрої повинні бути безпечними для використання;
- випромінювання екрану повинно бути зведено до незначного рівня;
- усі символи на екрані повинні бути чіткими та помітними;
- екран не повинен миготіти;
- працівник повинен мати змогу змінити яскравість екрану;
- екран не повинен відблискувати;

- клавіатура має бути відокремленою від екрану, щоб робітник мав змогу прийняти зручну позу для роботи;
- поверхня клавіатури теж не повинна відбивати світло, символи на клавіатурі повинні бути чіткими;
- програмне забезпечення екранного пристрою повинно бути простим у використанні для працівника.

Крім того існують й загальні положення про санітарні норми, правила та вимоги, які стосуються робочого процесу не лише програміста-розробника. Ці положення описані в наступних документах:

- Закон України від 14 жовтня 1992 р. № 2694-XII "Про охорону праці". Редакція від 16.10.2020 [61];
- Кодекс законів про працю України від 10 грудня 1971 р. № 322-VIII. Редакція від 25.10.2020 [62];
- Закон України від 23 вересня 1999 р. № 1105-XIV "Про загальнообов`язкове державне соціальне страхування". Редакція від 25.10.2020 [63];
- Постанова Кабінету Міністрів України від 01 серпня 1992 р. № 442 "Про затвердження Порядку проведення атестації робочих місць за умовами праці". Редакція від 28.10.2016 [64];
- Постанова Кабінету Міністрів України від 17.04.2019 №337 "Про затвердження Порядку розслідування та обліку нещасних випадків, професійних захворювань та аварій на виробництві" [65];
- НПАОП 0.00-4.12-05 Типове положення про порядок проведення навчання і перевірки знань з питань охорони праці, затверджене наказом Держнаглядохоронпраці від 26.01.2005 №15. Зареєстрованого в Мін'юсті України 15.02.2005 за №231/10511. Редакція від 14.04.2017 [66];
- Державні будівельні норми України «Природне і штучне освітлення» ДБН В.2.5-28:2018, затверджені наказом Міністерства регіонального розвитку, будівництва та житлово-комунального господарства України 03.10.2018 № 264, введено в дію з 01.03.2019 [67];

- ДСН 3.3.6.042-99 Державні санітарні норми мікроклімату виробничих приміщень, затверджені Постановою головного санітарного лікаря України №42 від 1 грудня 1999 року [68];
- НАПБ А.01.001-2014 Правила пожежної безпеки в Україні, затверджений наказом Міністерства внутрішніх справ України від 30.12.2014 №1417 і зареєстрований у Міністерстві юстиції України 05.03.2015 за № 252/26697 [69];
- Державні санітарні норми виробничої загальної та локальної вібрації ДСН 3.3.6.039-99, початок дії від 01.12.1999 [70];
- Кодекс цивільного захисту України. Редакція від 16.10.2020 [71].

5.2 Шкідливі виробничі фактори на робочому місці

До шкідливих виробничих факторів, які можуть негативно вплинути на програміста-розробника, можна віднести:

- незадовільний мікроклімат (температура, вологість, вентиляція повітря, інфрачервоне або ультрафіолетове випромінювання);
- постійні електричні поля або випромінювання;
- небезпечні іонізуючі випромінювання;
- високий рівень шумів та вібрацій;
- підвищена запиленість робочого місця;
- ризик отримати удар електричним струмом;
- напруження зорового апарату;
- недостатнє природне або технічне освітлення в робочих приміщеннях.

Рекомендації щодо їх усунення або максимального зниження їх впливу описані в підпунктах 5.2.1-5.2.4. Також в цих підпунктах було перевірено, чи задовольняє поточне робоче місце нормам та вимогам нормативних актів.

5.2.1 Мікроклімат приміщення

Мікроклімат – це клімат робочого приміщення, що визначається такими фізичними показниками:

- температура повітря;
- вологість повітря;
- швидкість руху повітря.

Щоб визначити, чи є фактичні показники оптимальними або допустимими, одразу потрібно з’ясувати, до якої категорії робіт відноситься розробка програми. Оскільки, ця робота потребує мінімум фізичної активності, цю роботу можна віднести до категорії «Легка 1а». Згідно з ДСН 3.3.6.042-99 [68] у табл. 5.1 встановлені оптимальні та допустимі показники мікроклімату у робочому приміщенні. Також у таблиці йде порівняння з фактичними показниками.

Таблиця 5.1 – Нормативні та фактичні величини клімату у робочому приміщенні

Період року	Параметри мікроклімату	Оптимальна величина	Допустима величина	Фактична величина
Холодний	Температура повітря в приміщенні	22 - 24°С	21 - 25° С	24,8°С
	Відносна вологість	40 - 60%	75%	48,8%
	Швидкість руху повітря	До 0.1 м/с	До 0.1 м/с	0,056 м/с
Теплий	Температура повітря в приміщенні	23 - 25° С	22 - 28° С	27,6°С
	Відносна вологість	40 - 60%	55% при 27 ° С	44,2%
	Швидкість руху повітря	До 0.1 м/с	До 0.1 – 0.2 м/с	0,096 м/с

З наведеної вище таблиці можна зробити висновок, що в цілому робоче приміщення, де розроблялося програмне забезпечення, відповідає визначеним нормам та вимогам. Показники відносної вологості та швидкості руху повітря є оптимальними, а температура повітря в приміщенні вже виходить за межі оптимальних температур. Фактична температура повітря в обидва періоди року трохи вище оптимальної, але все ще знаходиться в межах допустимої.

Щоб мати змогу підтримувати оптимальні або допустимі показники клімату необхідно дотримуватися таких рекомендацій:

- використовувати зволожувачі повітря;
- приміщення має бути обладнане системою вентиляції;
- у приміщенні повинен бути кондиціонер.

5.2.2 Шум та вібрації

Шум та вібрації є тими шкідливими виробничими факторами, тривалий вплив яких може негативно вплинути на здоров’я людини. Наслідками тривалого впливу можуть бути такі симптоми, як швидка втомлюваність, дратівливість, головні болі, психологічне напруження, зниження концентрації уваги, вушна біль та інше. Всі ці симптоми випливають у загальне зниження ефективності роботи працівника.

Щоб запобігти негативного впливу шуму та вібрацій слід дотримуватися таких рекомендацій:

- Стіни та стеля приміщення повинні бути облицьованими звуковбирними матеріалами;
- Потрібно використовувати м’які килимки.
- Можна використовувати навушники, вкладиші й таке інше.

На робочому місці програміста-розробника в якості джерела шуму та вібрацій може виступати персональний комп’ютер та його складові, а також кондиціонер. За допомогою шумоміру було виміряно показник шуму безпосередньо на робочому місці і він складає 36 дБ. Оскільки згідно з ДСН 3.3.6.039-99 показник шуму не повинен перевищувати 50дБ, можна впевнено сказати, що робоче місце відповідає вимогам по рівню шуму та вібрацій.

5.2.3 Освітлення робочого місця

Оптимальне освітлення робочих місць – одна з найважливіших вимог безпеки праці. Багаторазові спостереження за різними видами праці і спеціальні експерименти показали, що продуктивність праці людини крім багатьох виробничих факторів, залежить також і від освітленості робочого місця працівника. При недостатньому освітленні зорове сприйняття знижується, розвивається короткозорість та хвороби очей і головний біль. Через постійне зорове напруження настає зорова втома. При недостатньому освітленні працюючий близько нахиляється до обладнання, в результаті чого виникає небезпека нещасного випадку.

Залежно від джерела світла виробниче освітлення може бути: природним, що створюється прямими сонячними променями та розсіяним світлом небосхилу;

штучним, що створюється електричними джерелами світла; суміщеним, при якому недостатнє за нормами природне освітлення доповнюється штучним.

В свою чергу штучне освітлення теж можна поділити на такі категорії: загальне, місцеве та комбіноване. Слід зазначити, що застосовування лише місцевого освітлення не допускається з огляду на небезпеку виробничого травматизму та професійних захворювань.

Оскільки під час розробки програми постійно потрібно розпізнавати доволі малі об’єкти (0.3-0.5 мм), тобто символи на екрані, то цю роботу потрібно віднести до третього розряду (високої точності) зорової роботи. Це означає, що згідно ДБН В.2.5-28:2018 [67] загальна освітленість приміщення повинна дорівнювати 200 лк, а комбінована – 400 лк. За допомогою люксметру було визначено такі фактичні показники – 213.6 лк при загальному освітленні та 385.1 лк при комбінованому освітленні.

З отриманих показників можна зробити висновок, що фактичні показники освітленості наближені до нормативних, а значить відповідають вимогам, зазначеним у документі. Для освітлення робочого приміщення було використано природній та штучний джерела освітлення, які були встановлені відповідно до вимог.

5.3 Дії працівників в надзвичайних ситуаціях

Останніми роками у світі збільшилася кількість випадків захоплення заручників. Відомим терористичним актом останнього часу є захоплення в заручники глядачів та акторів мюзиклу ”Норд-Ост” у театральному центрі на Дубровці в Москві, де були громадяни України. Такі злочини – образа честі і гідності захоплених людей, заподіяння їм моральних і фізичних страждань, тілесних ушкоджень і навіть смерті.

Нижче наведено правила безпеки, яких важливо дотримуватися, якщо вас захопили злочинці як заручника:

- якщо злочинці в стані сп`яніння, то обмежуйте контакти з ними;
- не посилюйте агресивність злочинців непокорою, лайкою, зайвим опором;
- виконуйте вимоги терористів;
- уникайте будь-яких дискусій;

– з першою нагодою намагайтеся повідомити про своє місцезнаходження рідним чи в поліцію;

– намагайтеся запам`ятати будь-яку інформацію (вік, ріст, голос, манеру розмовляти, звички тощо) про злочинців;

– не дозволяйте собі падати духом;

– стежте за поведінкою терориста, намагайтеся втекти;

– при звільнені виходьте якомога швидше, речі залишайте там, де вони лежать, оскільки там можливі вибух або пожежа.

Висновки до розділу 5

У даному розділі були розглянути основні питання, які стосуються безпеки та охорони праці під час процесу розробки програмного забезпечення.

Були розглянуті основоположні документі, які визначають необхідні норми та вимоги до робочого процесу. Були визначені основні показники мікроклімату, вібрації, шуму та освітлення для конкретного робочого місця, на якому проводилася розробка.

Після їх визначення результати були співставленні з нормативними значеннями, описаними у відповідних документах. Виявилося, що в цілому отримані показники відповідають показникам, зазначеним у нормативних документах.

Також були розглянуті правила поведінки в надзвичайній ситуації, а саме – під час терористичного акту.

ВИСНОВКИ

Результатом даної дипломної роботи є визначення рівня впливу різноманітних показників на час роботи алгоритмів видалення невидимих ліній та поверхонь. Було досліджено вплив на час роботи HSR-алгоритмів таких показників:

- кількість полігонів тривимірних моделей;
- кількість пікселів, які потрібно обробити алгоритмом;
- роздільна здатність екрану або простору, на якому потрібно відобразити тривимірну модель.

Після детального аналізу предметної області було вирішено обирати алгоритми серед тих, які відносяться до алгоритмів, працюючих у просторі зображення, бо вони мають більшу ефективність, ніж алгоритми, працюючі у просторі об’єктів. Другі працюють з більшою точністю, але зараз для більшості сценаріїв точність відображення тривимірної комп’ютерної графіки не є пріоритетною, на відміну від продуктивності.

Для подальшого дослідження серед алгоритмів, працюючих у просторі зображення, було обрано такі:

- алгоритм Z-буфера;
- scan-line алгоритм;
- алгоритм «художника».

Було розглянуто зовнішнє і внутрішнє проектування програмної системи для збору даних та проведення досліджень. Визначені основні аспекти розробки користувацького інтерфейсу та розроблено інтуїтивно зрозумілий інтерфейс.

Для того, щоб мати змогу збирати дані часу роботи HSR-алгоритмів в визначених ситуаціях було розроблено програмний додаток. У ньому були реалізовані вищезгадані алгоритми. Він надає користувачу можливість визначати параметри експерименту, обирати HSR-алгоритм та тривимірну модель та отримувати результати експерименту в зручній формі (у файлі формату .CSV) для подальшого аналізу.

Для визначення впливу кожного з вищезгаданих показників на час роботи HSR-алгоритми було проведено три експерименти. Експерименти були поставлені таким

чином, щоб результати експериментів показували залежність часу роботи саме від одного конкретного показника.

Отримані дані експерименту було проаналізовано. На їх основі було визначено коефіцієнти впливу (див. табл. 4.7) вищезгаданих показників на той чи інший HSR-алгоритм. Ці коефіцієнти у числовій формі відображають, наскільки сильно впливає конкретний показник на час роботи алгоритму. В цілому отримані коефіцієнти кажуть про наступне:

− час роботи алгоритму «художника» дуже залежить від складності тривимірної моделі, а саме від кількості полігонів; з кожним новим полігоном час роботи цього алгоритму зростає на таку ж величину, що й з кожними 18 полігонами для алгоритму Z-буферу;

− час роботи усіх трьох досліджуваних алгоритмів зростає мінімально з підвищенням роздільної здатності екрану; час роботи алгоритмів зростає приблизно на від 4 до 5.3 мс (на пристрої, де було проведено експеримент) з кожним мільйоном пікселів (еквівалентно кількості пікселів на екрані з роздільною здатністю 1000x1000);

− показник кількості пікселів, які потрібно обробити алгоритмом, є найвпливовішим серед інших, не враховуючи алгоритм «художника» (тут найвпливовішим є саме кількість полігонів); цей показник менше всього впливає на час роботи алгоритму «художника»; для обробки кожного нового пікселя scan-line алгоритму в середньому потрібно на 49% більше часу, ніж попередньому HSR-алгоритму, а алгоритму Z-буфера – на 80% більше часу.

За допомогою знайдених коефіцієнтів можна визначити очікуваний час роботи алгоритму по вхідним даним. Їх використання дозволить обчислити час роботи кожного з алгоритмів та визначити найефективніший з них в конкретній ситуації безпосередньо перед запуском алгоритму.

Кожний з алгоритмів видалення невидимих ліній та поверхонь має свої сильні та слабкі сторони. Але на основі результатів цього дослідження, інші розробники можуть розробити метод, який буде приймати рішення, який алгоритм потрібно

використати, та таким чином досягти максимальної ефективності за рахунок того, що будуть використані сильні сторони кожного з HSR-алгоритмів.

Крім того, у дипломній роботі було розглянуто основні питання, які стосуються охорони праці та безпеки в надзвичайних ситуаціях на підприємствах згідно з нормативним актами та санітарними нормами. Зокрема було розглянуто наступні пункти:

- вимоги до безпеки на робочому місці;
- шкідливі виробничі фактори на робочому місці програміста-розробника;
- правила поведінки працівників під час терористичного акту.

БІБЛІОГРАФІЧНИЙ СПИСОК

1. «Hidden-surface determination» [Ел. ресурс]. Available: https://en.wikipedia.org/wiki/Hidden-surface_determination [Дата звернення: 30.11.2020].
2. «Классификация алгоритмов удаления» [Ел. ресурс]. Available: https://portal.tpu.ru/SHARED/a/AD/Education/Tab2/CG_hiding_surfaces.pdf [Дата звернення: 30.11.2020]
3. «Сравнительный анализ алгоритмов удаления невидимых линий и поверхностей, работающих в пространстве изображения» [Ел. ресурс]. Available: https://novainfo.ru/article/3958 [Дата звернення: 30.11.2020]
4. «Z-buffering» [Ел. ресурс]. Available: https://en.wikipedia.org/wiki/Z-buffering [Дата звернення: 30.11.2020]
5. «Z-fighting» [Ел. ресурс]. Available: https://en.wikipedia.org/wiki/Z-fighting [Дата звернення: 30.11.2020]
6. «Painter’s algorithm» [Ел. ресурс]. Available: https://en.wikipedia.org/wiki/Painter%27s_algorithm [Дата звернення: 30.11.2020]
7. «Computational Geometry - Algorithms and Applications» [Ел. ресурс]. Available: https://people.inf.elte.hu/fekete/algoritmusok_msc/terinfo_geom/konyvek/Computational%20Geometry%20-%20Algorithms%20and%20Applications,%203rd%20Ed.pdf [Дата звернення: 30.11.2020]
8. «A Solution to the Hidden Surface Problem» [Ел. ресурс]. Available: https://ohiostate.pressbooks.pub/app/uploads/sites/45/2017/09/newell-newell-sancha.pdf [Дата звернення: 30.11.2020]
9. «Newell's algorithm» [Ел. ресурс]. Available: https://en.wikipedia.org/wiki/Newell%27s_algorithm. [Дата звернення: 30.11.2020].
10. «Binary space partitioning» [Ел. ресурс]. Available: https://en.wikipedia.org/wiki/Binary_space_partitioning. [Дата звернення: 30.11.2020].

11. «Visible surface detection» [Ел. ресурс]. Available: https://www.tutorialspoint.com/computer_graphics/visible_surface_detection.htm. [Дата звернення: 01.12.2020].
12. «Warnock algorithm» [Ел. ресурс]. Available: https://en.wikipedia.org/wiki/Warnock_algorithm. [Дата звернення: 01.12.2020].
13. «Rasterisation» [Ел. ресурс]. Available: https://en.wikipedia.org/wiki/Rasterisation. [Дата звернення: 01.12.2020].
14. «Polygon mesh» [Ел. ресурс]. Available: https://en.wikipedia.org/wiki/Polygon_mesh. [Дата звернення: 01.12.2020].
15. «STL (file format)» [Ел. ресурс]. Available: https://en.wikipedia.org/wiki/STL_(file_format). [Дата звернення: 01.12.2020].
16. «The Stl format» [Ел. ресурс]. Available: http://www.fabbers.com/tech/STL_Format. [Дата звернення: 01.12.2020].
17. «STL File Format (3D Printing) – Simply Explained» [Ел. ресурс]. Available: https://all3dp.com/what-is-stl-file-format-extension-3d-printing/. [Дата звернення: 01.12.2020].
18. «Back-face culling» [Ел. ресурс]. Available: https://www.computerhope.com/jargon/b/backface-culling.htm. [Дата звернення: 01.12.2020].
19. «Hidden surface removal» [Ел. ресурс]. Available: https://dondi.lmu.build/share/cg/hsr-v02.pdf. [Дата звернення: 01.12.2020].
20. «Материалы для изучения Autodesk Meshmixer» [Ел. ресурс]. Available: https://mikhailov-andrey-s.blogspot.com/2017/07/materialy-dlya-izucheniya-autodesk-meshmixer.html. [Дата звернення: 01.12.2020].
21. «Статистика разрешений экранов с 2000-2017 год» [Ел. ресурс]. Available: https://fortress-design.com/display-resolution/. [Дата звернення: 02.12.2020].
22. «Graphics display resolution» [Ел. ресурс]. Available: https://en.wikipedia.org/wiki/Graphics_display_resolution. [Дата звернення: 02.12.2020].

23. «Time complexity» [Ел. ресурс]. Available: https://en.wikipedia.org/wiki/Time_complexity. [Дата звернення: 02.12.2020].
24. «Коефіцієнт детермінації» [Ел. ресурс]. Available: https://uk.wikipedia.org/wiki/Коефіцієнт_детермінації. [Дата звернення: 02.12.2020].
25. «Довірчий інтервал» [Ел. ресурс]. Available: https://uk.wikipedia.org/wiki/Довірчий_інтервал. [Дата звернення: 02.12.2020].
26. «Діаграма прецедентів» [Ел. ресурс]. Available: https://uk.wikipedia.org/wiki/Діаграма_прецедентів. [Дата звернення: 03.12.2020].
27. «Природність інтерфейсу» [Ел. ресурс]. Available: https://wiki.cuspu.edu.ua/index.php/Природність_інтерфейсу. [Дата звернення: 03.12.2020].
28. «Проектування користувацького інтерфейсу» [Ел. ресурс]. Available: http://moodle.ipo.kpi.ua/moodle/mod/resource/view.php?id=39243. [Дата звернення: 03.12.2020].
29. «Правило Миллера» [Ел. ресурс]. Available: https://psyfactor.org/personal/personal15-20.htm. [Дата звернення: 03.12.2020].
30. «Проектирование графического интерфейса пользователя» [Ел. ресурс]. Available: https://habr.com/ru/post/208966/. [Дата звернення: 03.12.2020].
31. «Sequence diagram» [Ел. ресурс]. Available: https://en.wikipedia.org/wiki/Sequence_diagram. [Дата звернення: 03.12.2020].
32. «Unity (game engine)» [Ел. ресурс]. Available: https://en.wikipedia.org/wiki/Unity_(game_engine). [Дата звернення: 03.12.2020].
33. «Краткий обзор C#» [Ел. ресурс]. Available: https://docs.microsoft.com/ru-ru/dotnet/csharp/tour-of-csharp/. [Дата звернення: 03.12.2020].
34. «Сборка мусора» [Ел. ресурс]. Available: https://docs.microsoft.com/ru-ru/dotnet/standard/garbage-collection/. [Дата звернення: 03.12.2020].

35. «Исключения и обработка исключений» [Ел. ресурс]. Available: https://docs.microsoft.com/ru-ru/dotnet/csharp/programming-guide/exceptions/. [Дата звернення: 03.12.2020].
36. «Лямбда-выражения» [Ел. ресурс]. Available: https://docs.microsoft.com/ru-ru/dotnet/csharp/language-reference/operators/lambda-expressions. [Дата звернення: 03.12.2020].
37. «Синтаксис LINQ» [Ел. ресурс]. Available: https://docs.microsoft.com/ru-ru/dotnet/csharp/linq/. [Дата звернення: 03.12.2020].
38. «Асинхронное программирование с использованием ключевых слов async и await» [Ел. ресурс]. Available: https://docs.microsoft.com/ru-ru/dotnet/csharp/programming-guide/concepts/async/. [Дата звернення: 03.12.2020].
39. «Сопоставление шаблонов» [Ел. ресурс]. Available: https://docs.microsoft.com/ru-ru/dotnet/csharp/pattern-matching/. [Дата звернення: 03.12.2020].
40. «Типы» [Ел. ресурс]. Available: https://docs.microsoft.com/ru-ru/dotnet/csharp/programming-guide/types/. [Дата звернення: 03.12.2020].
41. «Система управления версиями» [Ел. ресурс]. Available: https://ru.wikipedia.org/wiki/Система_управления_версиями. [Дата звернення: 03.12.2020].
42. «About: Branching and Merging» [Ел. ресурс]. Available: https://git-scm.com/about/branching-and-merging. [Дата звернення: 03.12.2020].
43. «About: Small and Fast» [Ел. ресурс]. Available: https://git-scm.com/about/small-and-fast. [Дата звернення: 03.12.2020].
44. «YAGNI principle» [Ел. ресурс]. Available: https://whatis.techtarget.com/definition/You-arent-gonna-need-it. [Дата звернення: 04.12.2020].
45. «Don’t repeat yourself» [Ел. ресурс]. Available: https://deviq.com/don-t-repeat-yourself/. [Дата звернення: 04.12.2020].

46. «DRY or WET and Why?» [Ел. ресурс]. Available: https://medium.com/@nrk25693/dry-or-wet-and-why-867ac3096483. [Дата звернення: 04.12.2020].
47. «Концепция: Стратегия тестирования» [Ел. ресурс]. Available: http://dit.isuct.ru/Publish_RUP/core.base_rup/guidances/concepts/test_strategy_9981F03E.html. [Дата звернення: 04.12.2020].
48. «Тестирование по стратегии чёрного ящика» [Ел. ресурс]. Available: https://ru.wikipedia.org/wiki/Тестирование_по_стратегии_чёрного_ящика. [Дата звернення: 04.12.2020].
49. «Тестирование по белого ящика» [Ел. ресурс]. Available: https://ru.wikipedia.org/wiki/Тестирование_белого_ящика. [Дата звернення: 04.12.2020].
50. «View the call stack and use the Call Stack window in the debugger» [Ел. ресурс]. Available: https://docs.microsoft.com/en-us/visualstudio/debugger/how-to-use-the-call-stack-window?view=vs-2019. [Дата звернення: 04.12.2020].
51. «Use breakpoints in the Visual Studio debugger» [Ел. ресурс]. Available: https://docs.microsoft.com/en-us/visualstudio/debugger/using-breakpoints?view=vs-2019. [Дата звернення: 04.12.2020].
52. «Debug.Log» [Ел. ресурс]. Available: https://docs.unity3d.com/ScriptReference/Debug.Log.html. [Дата звернення: 04.12.2020].
53. «Stopwatch Class» [Ел. ресурс]. Available: https://docs.microsoft.com/en-us/dotnet/api/system.diagnostics.stopwatch?view=net-5.0. [Дата звернення: 05.12.2020].
54. «Сборщик мусора в C#» [Ел. ресурс]. Available: https://metanit.com/sharp/tutorial/8.1.php. [Дата звернення: 05.12.2020].
55. «GC.Collect Method» [Ел. ресурс]. Available: https://docs.microsoft.com/en-us/dotnet/api/system.gc.collect?view=net-5.0. [Дата звернення: 05.12.2020].
56. НПАОП 0.700-7.15-18 Вимоги щодо безпеки та захисту здоров’я працівників під час роботи з екранними пристроями, затверджені від 14.02.2018 р. № 207.

57. НПАОП 80.0-1.12-04 Правила безпеки під час навчання в кабінетах інформатики навчальних закладів системи загальної середньої освіти, затверджені від 20.11.2007 р. № 252.
58. ДСанПіН 3.3.2-007-98 Державні санітарні правила і норми роботи з візуальними дисплейними терміналами електронно-обчислювальних машин, затверджені від 10.12.1998 р. № 7.
59. Інструкція щодо надання послуг з ремонту побутової радіоелектронної апаратури, затверджена від 27.08.2000 р. № 20.
60. ДСН 3.3.6.042-99, Державні санітарні норми мікроклімату виробничих приміщень затверджені від 01.12.99 р. № 42.
61. Закон України “Про охорону праці” прийнятий від 14 жовтня 1992 року № 2695-12.
62. Кодекс законів про працю України від 10 грудня 1971 р. № 322-VIII. Редакція від 25.10.2020.
63. Закон України від 23 вересня 1999 р. № 1105-XIV “Про загальнообов`язкове державне соціальне страхування”. Редакція від 25.10.2020.
64. Постанова Кабінету Міністрів України від 01 серпня 1992 р. № 442 “Про затвердження Порядку проведення атестації робочих місць за умовами праці”. Редакція від 28.10.2016.
65. Постанова Кабінету Міністрів України від 17.04.2019 №337 “Про затвердження Порядку розслідування та обліку нещасних випадків, професійних захворювань та аварій на виробництві”.
66. НПАОП 0.00-4.12-05 Типове положення про порядок проведення навчання і перевірки знань з питань охорони праці, затверджене наказом Держнаглядохоронпраці від 26.01.2005 №15. Зареєстрованого в Мін’юсті України 15.02.2005 за №231/10511. Редакція від 14.04.2017.
67. Державні будівельні норми України «Природне і штучне освітлення» ДБН В.2.5-28:2018, затверджені наказом Міністерства регіонального розвитку, будівництва та житлово-комунального господарства України 03.10.2018 № 264, введено в дію з 01.03.2019.

68. ДСН 3.3.6.042-99 Державні санітарні норми мікроклімату виробничих приміщень, затверджені Постановою головного санітарного лікаря України №42 від 1 грудня 1999 року.
69. НАПБ А.01.001-2014 Правила пожежної безпеки в Україні, затверджений наказом Міністерства внутрішніх справ України від 30.12.2014 №1417 і зареєстрований у Міністерстві юстиції України 05.03.2015 за № 252/26697.
70. Державні санітарні норми виробничої загальної та локальної вібрації ДСН 3.3.6.039-99, початок дії від 01.12.1999.
71. Кодекс цивільного захисту України від 02 жовтня 2012 року № 5403-VI. Редакція від 16.10.2020.

ДОДАТКИ

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

ЗАТВЕРДЖУЮ
Перший проректор Дніпровського
національного університету
залізничного транспорту
імені академіка В. Лазаряна
Б.Є. Боднар

СИСТЕМА ДЛЯ ДОСЛІДЖЕННЯ ЧАСОВОЇ ЕФЕКТИВНОСТІ АЛГОРИТМІВ ВИДАЛЕННЯ НЕВИДИМИХ ЛІНІЙ ТА ПОВЕРХОНЬ

Технічне завдання
ЛИСТ ЗАТВЕРДЖЕННЯ
1116130.01193-01-ЛЗ

Завідувач кафедри КІТ
проф. В.І. Шинкаренко

Керівник розробки
доц. О.П. Іванов

Виконавець
студент групи ПЗ1921
І.О. Строчіков

Нормоконтролер
доц. О.С. Куроп'ятник

2020

ЗАТВЕРДЖЕНО
1116130.01193-01-ЛЗ

СИСТЕМА ДЛЯ ДОСЛІДЖЕННЯ ЧАСОВОЇ ЕФЕКТИВНОСТІ АЛГОРИТМІВ ВИДАЛЕННЯ НЕВИДИМИХ ЛІНІЙ ТА ПОВЕРХОНЬ

1116130.01193-01

Аркушів 23

2020

АНОТАЦІЯ

Документ 1116130.01193-01 «Система для дослідження часової ефективності алгоритмів видалення невидимих ліній та поверхонь. Технічне завдання» входить до складу програмної документації до дипломного проекту.

У даному документі представлене призначення та область застосування програмного продукту, основні вимоги, стадії та строки виконання проекту, техніко-економічні показники, що пред’являються до програмного продукту.

ЗМІСТ

ВСТУП

В наш час вже більшість людей так чи інакше стикаються з комп’ютерною графікою, бо у кожного є власний смартфон або персональний комп’ютер. Для комфортного використання програм для моделювання або навіть ігор з різноманітною графікою дуже важливо аби об’єкти на сцені відмальовувались максимально швидко з прийнятною частотою (40-60 кадрів в секунду). Це особливо важливо для мобільних пристроїв, які є дуже популярними зараз, але водночас ще дуже поступаються персональним комп’ютерам в обчислювальній потужності.

Тому варто звернути увагу на часову ефективність алгоритмів видалення невидимих ліній та поверхонь, бо вони займають доволі велику частку загального часу рендеру об’єктів. Як загально відомо – кожен алгоритм видалення невидимих ліній та поверхонь має свої переваги та недоліки, тому є сенс дослідити, які з них працюють ефективніше в тій чи іншій ситуації.

Ці алгоритми мають єдину ціль – якомога швидше вирішити, які з вхідних пікселів потрібно відобразити, а які – ні. Це робиться на основі даних про кожний піксель. В загальному випадку просто порівнюються так звані z-координати (дальність від погляду користувача) кожного пікселя один з одним. А кожний алгоритм – це конкретна реалізація цього порівняння, яка може опинитися більш або менш вдалою для деякої ситуації.

Задача, яку вирішують алгоритми видалення невидимих ліній та поверхонь відносяться до задач м’якого реального часу, бо для користувача хоч і не бажано, але припустимо, коли час вирішення задачі буде перевищувати допустимий. Це може трапитися в тому випадку, коли сцена, що відображається, є дуже складною і неоптимізованою для поточного пристрою для відображення. Такі випадки приводять до того, що частота рендеру сцени стає меншою, через що користувач може помітити сіпання на екрані або навіть повністю статичну сцену з рідкою зміною кадрів приблизно 1 у секунду. Через це глядач не зможе належним чином зануритися у процес і нормально реагувати на побачене.

На сьогоднішній день існує безліч алгоритмів видалення невидимих ліній та поверхонь, але на практиці використовуються лише деякі з них. Це ті алгоритми, які

показують найкращі результати для більшості ситуацій. Наприклад, в широко відомій графічній бібліотеці OpenGL реалізовано лише алгоритм Z-буфера. І користувач бібліотеки зможе використати лише цей алгоритм для видалення невидимих ліній та поверхонь. І хоча цей алгоритм зможе опрацювати графічну сцену абсолютно будь-якої складності, це ще не означає, що він є найефективнішим в будь-якому сценарії.

До недавніх часів це не було серйозною проблемою, бо навіть якщо обраний алгоритм буде працювати трохи повільніше іншого, то різниця буде настільки малою, що користувач просто цього не помітить, а частота кадрів в секунду буде все ще прийнятною. Але зараз з'явилися мобільні гаджети, які мають набагато меншу обчислювальну потужність, аніж персональні комп'ютери. Тим паче вони мають меншу кількість оперативної пам'яті, що може позначитися на ефективності алгоритмів, яким для роботи якраз потрібно багато пам'яті.

З розширенням технічних можливостей пристроїв, виросли й вимоги до комп'ютерної графіки. Користувачі почали звикати до поточного стану, коли графіка, що відображається, виглядає дуже реалістичною за рахунок більш різноманітної палітри кольорів та плавної анімації. Але з появою мобільних пристроїв виявилося, що досягти такого ж результату й тут неможливо, тому розробникам мобільних додатків та ігор часто доводиться зменшувати якість комп'ютерної графіки або витрачати велику кількість часу та ресурсів на оптимізацію задля більшої продуктивності.

1 ПІДСТАВА ДО РОЗРОБКИ

Підставою для розробки даного програмного продукту є наказ №779 ст. «Про призначення керівників та затвердження тем магістерських робіт» від 10.10.2019 р., затверджений ректором Дніпровського національного університету залізничного транспорту імені академіка В. Лазаряна.

Тема: Дослідження часової ефективності алгоритмів комп’ютерної графіки для видалення невидимих ліній та поверхонь.

Керівник дипломного проекту – доц. Іванов О.П.

2 ПРИЗНАЧЕННЯ РОЗРОБКИ

2.1 Функціональне призначення

За допомогою даного продукту користувач матиме змогу проводити експерименти для оцінки часової ефективності таких алгоритмів видалення невидимих ліній та поверхонь: Z-буферу, scan-line, «художника». Використовуючи різні комбінації вхідних даних, він зможе збирати результати експериментів для подальшого аналізу.

2.2 Експлуатаційне призначення

Програмний продукт призначений для забезпечення швидкого збору даних щодо часу роботи вищезгаданих алгоритмів видалення невидимих ліній та поверхонь в різноманітних умовах.

3 ВИМОГИ ДО ПРОГРАМИ

3.1 Вимоги до функціональних характеристик

Програма повинна:

- мати можливість працювати з файлами формату .STL;
- надавати можливість ввести параметри експерименту (роздільна здатність, кількість тестів, тривимірна модель, HSR-алгоритм);
- обробляти тривимірну модель обраним HSR-алгоритмом;
- зберігати часові результати експерименту у файл;
- відображати оброблену тривимірну модель.

Вхідні дані:

- ширина та висота зображення (цілі числа);
- кількість тестів (ціле число);
- унікальний номер HSR-алгоритму, який буде використано (ціле число);
- файл формату .STL, який потрібно обробити (текст).

Вихідні дані:

- візуальне відображення обраної тривимірної моделі;
- файл формату .CSV, який містить результати експерименту та має такі дані:
    e. використаний HSR-алгоритм;
    f. роздільна здатність екрану;
    g. кількість полігонів моделі;
    h. час роботи алгоритму.

3.2 Вимоги до надійності

Одним із критеріїв правильного функціонування програмного продукту є забезпечення надійності роботи програмного продукту. Вимоги до надійності програмного продукту повинні відповідати наступним вимогам:

- програма не повинна допускати невимушену втрату та пошкодження даних;

– кількість відмов системи не повинна перевищувати однієї відмови на 2000 запусків системи (під відмовою слід вважати непрацездатність системи після її запуску, тобто необхідність запустити систему повторно).

3.3 Умови експлуатації

Для нормального функціонування програмного продукту необхідно виконання наступних вимог:

– ЕОМ повинні відповідати вимогам чинних в Україні стандартів, нормативних актів з охорони праці [1];
– програмний комплекс повинен використовуватись в приміщеннях, призначених для роботи ЕОМ з наступними кліматичними умовами [2] температура – 21-25 °С, відносна вологість повітря 40-60%;
– користувач повинен бути ознайомлений з керівництвом користувача.

3.4 Вимоги до складу та параметрів технічних засобів

Для коректного функціонування програмного продукту вимагається наявність ЕОМ, що задовольняє нормальну роботу:

Мінімальна конфігурація комп’ютера для забезпечення роботи програмного продукту:

– ОС Windows 10;
– ОЗП не менш ніж 2048 МБ;
– вільний простір пам’яті не менше 300 МБ.

3.5 Вимоги до інформаційної та програмної сумісності

Для функціонування програмного продукту необхідна ОС Windows 10.

3.6 Вимоги до маркування та упаковки

Програма може зберігатися на змінних носіях (flash-памя’ть). Упаковка продукту повинна забезпечувати захист від механічних пошкоджень. Упаковка повинна мати маркування:

«Система для дослідження часової ефективності алгоритмів видалення невидимих ліній та поверхонь».

Розробник: Строчіков І.О.

Версія 1.0

ДНУЗТ, КІТ

2020

3.7 Вимоги до транспортування та зберігання

Транспортування може виконуватись будь-яким способом, що виключає механічний і електромагнітний вплив на носії інформації. Місце збереження повинне відповідати умовам збереження носія, на якому знаходиться програмний комплекс.

Термін збереження обумовлений збереженням інформації на носії. Рекомендується проводити профілактичні роботи з перевірки якості носіїв кожні шість місяців.

4 ВИМОГИ ДО ПРОГРАМНОЇ ДОКУМЕНТАЦІЇ

Програмна документація повинна складатися з двох частин:

– технічного завдання;

– робочого проекту.

Робочий проект повинен складатися з:

– специфікації;

– тексту програми;

– опису програми;

– опису застосування;

– керівництва користувача. Керівництво по роботі з системою для дослідження часової ефективності алгоритмів видалення невидимих ліній та поверхонь.

Вся документація до програми повинна задовольняти вимогам державного стандарту до оформлення програмних документів ГОСТ 19.101-77 «Єдина система програмної документації. Види програм та програмних документів» [3].

5 ВИЗНАЧЕННЯ ВИТРАТ НА ПРОЕКТУВАННЯ ПРОГРАМИ

Капітальні вкладення включають всі витрати, пов’язані з реалізацією проекту. До них відносяться витрати на придбання технологій і обладнання, доставку на підприємство, монтаж і налагодження, на розробку проекту по заміні технологій і обладнання. Якщо у підприємства не вистачає власних коштів на цю заміну, воно може взяти кредит в банку. В цьому випадку капітальні витрати називаються інвестиціями.

До інших капітальних вкладень відносяться, зокрема витрати на технічні заходи з охорони навколишнього середовища, покращення умов праці. До них включається також залишкова вартість не повністю зношеного обладнання, що ліквідується.

Нова техніка, технологія, засоби автоматизації, що розробляються і впроваджуються у виробництво, повинні приносити певний корисний результат – ефект. Ефект може проявлятися у поліпшенні умов праці працюючих (соціальний), в зниженні шкідливого впливу виробництва на навколишнє середовище (екологічний), у підвищенні безпеки держави (оборонний), та, врешті, в економії витрат підприємства на виробництво продукції та збільшенні його прибутку (економічний).

Згідно моделі COCOMO, розмір проекту S вимірюється в рядках коду LOC (KLOC), а трудовитрати в людино-місяцях [4][5].

$$E = a \cdot S\hat{}b \cdot EAF, \tag{5.1}$$

де E – витрати праці на проект (в людино-місяцях);

S^b – розмір коду (в KLOC);

EAF – фактор уточнення витрат (effort adjustment factor).

Для простих систем, a = 2,4; b = 1,05

Припустимо, що розмір програмного коду програмного засобу – 900 рядків:

$$E = 2{,}4 \cdot 0{,}9^{1{,}05} \cdot 1 = 2{,}27. \tag{5.2}$$

Отже, згідно моделі COCOMO, орієнтовні трудовитрати на проект складуть приблизно 2 людино-місяці, що складає 40 робочих днів або 8 робочих тижнів, 4 робочі тижні на місяць, 5 робочих днів на тиждень, 8 робочих годин на день, одним виконавцем:

$$N_{\text{чол}} = 1;\ N_{\text{міс}} = 2; N_{\text{тиж}} = 8; N_{\text{днів}} = 40; Z_{\text{тиж}} = 4; Z_{\text{днів}} = 5; Z_{\text{год}} = 8. \quad (5.3)$$

Основними статтями витрат прийняті:

– основна заробітна плата;

– відрахування на соціальні потреби;

– накладні витрати;

– витрати на персональний комп'ютер і ліцензійні базові програмні засоби.

Основна заробітна плата (ОЗП) оцінює працю інженера-програміста зі створення програмного продукту і визначається виходячи з кількості розробників, часу виконання розробки та заробітної плати.

Таблиця 5.1 – Середньомісячна заробітна плата працівників у сфері інформації та телекомунікаційних технологій за період з січня 2020р. до вересня 2020р. за даними довідкової інформаційної Державної служби статистики України [6].

Рік	Місяць	Середня заробітна плата, грн
2020 Кількість місяців: 10	Січень	23597,00
	Лютий	24864,00
	Березень	33141,00
	Квітень	24109,00
	Травень	24304,00
	Червень	24681,00
	Липень	25922,00
	Серпень	26059,00
	Вересень	26329,00
	Жовтень	23597,00

Відповідно до даних наведених в табл. 5.1. можна розрахувати суму зарплат інженера-програміста за 10 місяців:

$$S_{\text{зарпл}} = \sum_{i=1}^{10} \text{Середня заробітна плата за і–й місяць} \quad (5.4)$$

$S_{зарпл}$= 23597,00 + 24864,00 + 33141,00 + 24109,00 + 24304,00 + 24681,00 + 25922,00 + 26059,00 + 26329,00 + 26239,00 = 259245,00 (5.5)

Розрахунок середньомісячної заробітної плати інженера-програміста за останні 10 місяців:

$$S_{міс} = \frac{S_{зарпл}}{N_{stat}}; \quad (5.6)$$

$$S_{міс} = \frac{259245,00}{10} = 25924,50 \text{ грн/міс}. \quad (5.7)$$

Середня погодинна заробітна плата:

$$N_{год} = \frac{S_{міс}}{Z_{тиж} \cdot Z_{днів} \cdot Z_{год}}; \quad (5.8)$$

$$N_{год} = \frac{25924,50}{4 \cdot 5 \cdot 8} = 162 \text{ грн/год}. \quad (5.9)$$

Описаний в проекті програмний продукт буде розроблений одним програмістом в період з 12.10.20 до 04.12.20, що складає 40 робочих днів або 8 робочих тижнів. Витрати робочого часу прийняті за 40 годин у тиждень. Погодинна ставка кваліфікованого інженера–програміста складає 162 грн/год. Таким чином, витрачено робочого часу:

$$t_{розробки} = N_{чол} \cdot N_{тиж} \cdot N_{год}, \quad (5.10)$$

де $N_{чол}$ – кількість виконавців, чол;

$N_{тиж}$ – тривалість розробки;

$N_{год}$ – витрати робочого часу, год.

$$t_{розробки} = 1 \cdot 8 \cdot 40 = 320 \text{ чол/год}, \quad (5.11)$$

ОЗП визначається за формулою:

$$ОЗП = t_{розробки} \cdot N \cdot K_{КВ}, \quad (5.12)$$

де $t_{розробки}$ – витрати праці у чол/год;

N – погодинна ставка;

$K_{КВ}$ – коефіцієнт кваліфікації програміста, приймається 0,75.

ОЗП складає:

$$ОЗП = 320 \cdot 162 \cdot 0,75 = 38880 \text{ грн}. \quad (5.13)$$

Відповідно до чинного Законодавства України нарахування на заробітну плату у вигляді Єдиного внеску на загальнообов'язкове державне соціальне страхування (ЄСВ) становить 22% від окладу працівника [7]. Таким чином ФОП з нарахуванням становить:

$$С_{соц} = \frac{ОЗП \cdot 22\%}{100\%}, \quad (5.14)$$

$$С_{соц} = \frac{38880 \cdot 22\%}{100\%} = 8554 \text{ грн}, \quad (5.15)$$

Отримані результати за (13) та (15) підсумовуються.

$$ОПВ = ОЗП + С_{соц} = 38880 + 8554 = 47434 \text{ грн.} \quad (5.16)$$

Вони визначають основні прямі витрати. Накладні витрати враховують загально господарчі витрати по забезпеченню проведення роботи: витрати на опалення, електроенергію, амортизацію будівель, зарплату адміністративного персоналу та інше. Вони становлять 30-40 % від суми прямих витрат на оплату праці:

$$С_{накл} = \frac{ОЗП \cdot 35\%}{100\%}, \quad (5.17)$$

$$С_{накл} = \frac{38880 \cdot 35\%}{100\%} = 13608 \text{ грн.}, \quad (5.18)$$

Протягом усього терміну використання нової техніки підприємство щорічно витрачає певні кошти, пов’язані з її експлуатацією.

Накладні витрати на проект визначаються терміном розробки програмної системи в залежності від вартості комп’ютеру та інших складових і включають в себе:

- витрати на електроенергію;
- вартість витратних матеріалів;
- заробітна плата ремонтника;
- амортизаційні витрати на комп’ютерне обладнання і програмне забезпечення;
- додаткові витрати – прибирання приміщення, охорона, оренда, комунальні послуги.

Розрахунок витрат на електроенергію

Витрати на електроенергію ($С_{ел}$) визначаються за формулою:

$$С_{ел} = Р \cdot В \cdot Т_{розр}, \quad (5.19)$$

де Р – потужність комп’ютера та допоміжних споживачів електричної енергії, приймаємо 0,4 *кВт/год;*

В – вартість 1 *кВт* складає 1,68 грн [8].

$Т_{розр}$ – час роботи з ЕВМ, прийнято рівним робочому часу, розраховується за формулою:

$$С_{ел} = 0{,}4 \cdot 1{,}68 \cdot 320 = 215 \text{ грн}, \quad (5.20)$$

Витрати на витратні матеріали ($С_{вм}$) протягом всього терміну експлуатації приблизно 10% від вартості комп’ютеру. Вартість робочої станції приймається 20 000 грн., термін експлуатації – 5 років. За робочу станцію приймається ноутбук HP Pavilion Gaming 15 [9]. Отже, можна визначити ці витрати за період створення програмного засобу:

$$С_{вм} = В_{ком} \cdot \frac{N_Д}{N_{експ} \cdot 365} \cdot \frac{10\%}{100\%}, \quad (5.21)$$

де $В_{ком}$ – вартість персонального комп’ютеру;

$N_Д$ – кількість днів розробки програмного продукту;

$N_{експ}$ – термін експлуатації персонального комп’ютеру.

Витрати на витратні матеріали визначаються так:

$$С_{вм} = 20000 \cdot \frac{40}{5 \cdot 365} \cdot \frac{10\%}{100\%} = 44 \text{ грн} \quad (5.22)$$

Заробітна плата ремонтника ($C_{рем}$) визначена наступним чином: на ремонт 50 комп’ютерів потрібен один інженер-системотехнік. Його середньомісячна заробітна плата приймається 15000 грн [10]. Тоді в перерахунку на один комп’ютер його заробітна плата складає:

$$C_{рем} = \frac{C_{рем}'}{N_{КОМ}} = \frac{15000}{50} \cdot 2 = 600, \quad (5.23)$$

де $C_{рем}'$ – середньомісячна заробітна плата;

$N_{КОМ}$ – кількість комп’ютерів на одного ремонтника.

За статистикою витрати на комплектуючи вироби ($С_{ком}$) для ремонту персонального комп’ютера складає 10% від його вартості за термін його експлуатації, тобто рівні витратам на витратні матеріали:

$$C_{\text{ком}} = \text{С}_{\text{вм}} = 44 \text{ грн}. \quad (5.24)$$

Амортизаційні відрахування на персональний комп'ютер (АПК) визначені з положення, що амортизаційний період в даний час дорівнює терміну морального старіння обчислювальної техніки і складає 5 років. Отже, за 5 років амортизаційні відрахування на персональний комп'ютер дорівнюють вартості комп'ютера. За період проектування амортизаційні відрахування складуть:

$$\text{АКП} = \text{В}_{КОМ} \cdot \frac{N_{\text{міс}}}{N_{\text{експ}} \cdot 12}, \quad (5.25)$$

$$\text{АКП} = 20000 \cdot \frac{2}{5 \cdot 12} = 667 \text{ грн}. \quad (5.26)$$

Амортизаційні відрахування на програмне забезпечення (АПЗ) залежать від його циклу заміни. Якщо прийняти термін морального старіння для Windows 5 років та Visual Studio за 1 рік то амортизаційні відрахування на програмне забезпечення дорівнюють його вартості.

Для функціонування персонального комп'ютера використовувалася операційна система Windows 10, для написання програмного забезпечення – програмне середовище Visual Studio 2019 та Unity 2019 Personal.

Розрахунок амортизаційних витрат на програмне забезпечення (АВП) приведений в табл. 5.2.

Таблиця 5.2 – Програмне забезпечення, що використовується в проекті

Найменування програмного забезпечення	Кількість, *шт*	Вартість програмного забезпечення, *грн*	Джерело придбання	Амортизаційні витрати, *грн*
Windows 10 Pro	1	6633 [11]	microsoft.com	221
Visual Studio 2019	1	14608 [12]	visualstudio.com	2435
Unity 2019 Personal	1	0	unity.com	0
Всього:	3	21241		2656

$$АКП_w = 6633 \cdot \frac{2}{5 \cdot 12} = 221 грн, \quad (5.27)$$

$$АКП_{vs} = 14608 \cdot \frac{2}{1 \cdot 12} = 2435 \text{ грн} \quad (5.28)$$

В результаті отримали суму амортизаційних витрат на програмне забезпечення (АПО), що дорівнює 2656 грн.

Додаткові витрати ($С_{дод}$) є індивідуальними в залежності від кожного проекту та визначаються окремо. Тому приймемо суму витрат у розмірі 50 % від заробітної плати інженера-програміста, що становить 12962 грн.

Оренду приміщень приймемо рівною 1960 гривень на місяць. Вартість оренди взята з реальної пропозиції [13]. Отже, оренда приміщення на 2 місяці становить 3920 грн.

Сумарні експлуатаційні витрати на один персональний комп'ютер складають:

$$C_{експ} = C_{ел} + C_{вм} + C_{рем} + АКП + АПО + C_{ор} + C_{дод}; \quad (5.29)$$

$$C_{експ} = 215 + 44 + 600 + 667 + 2656 + 3920 + 12962 = 21064 \text{ грн} \quad (5.30)$$

Результати розрахунків зведено у табл. 5.3.

Таблиця 5.3 – Експлуатаційні витрати на ПК і ПЗ.

Найменування витрат	Витрати, грн
Витрати на електроенергію	215
Вартість витратних матеріалів	44
Витрати на ремонт	600
Амортизація персонального комп'ютера	667
Амортизація програмного забезпечення	2656
Оренда приміщення	3920
Додаткові витрати	12962
Всього	21064

Таким чином, витрати на створення програмного продукту складають:

$$C_{\text{розробки}} = \text{ОЗП} + C_{\text{соц}} + C_{\text{накл}} + C_{\text{експ}}; \quad (5.31)$$

$$C_{\text{розробки}} = 38880 + 8554 + 13608 + 21064 = 82106 \text{ грн} \quad (5.32)$$

Розрахунок витрат зведено у табл. 5.4.

Таблиця 5.4 – Кошторис витрат на розробку програмного засобу

Найменування витрат	Витрати, грн
Основна заробітна плата	38880
Відрахування на соціальні потреби	8554
Накладні витрати	13608
Експлуатаційні витрати	21064
Всього	82106

За отриманими значеннями техніко-економічних показників проекту складено кошторис витрат на розроблення системи дослідження часової ефективності алгоритмів видалення невидимих ліній та поверхонь.

6 СТАДІЇ ТА ЕТАПИ РОЗРОБКИ

Стадії та етапи розробки програми представлені у табл. 6.1.

Таблиця 6.1 – Стадії та етапи розробки

Стадії розробки	Етапи розробки	Терміни виконання
1.Технічне завдання (ТЗ)	Постановка задачі	12.10.2020 – 13.10.2020
	Огляд літератури та аналіз аналогів	14.10.2020 – 16.10.2020
	Розробка структур вхідних і вихідних даних	19.10.2020 – 23.10.2020
	Визначення вимог до програми. Вибір та обґрунтування мови програмування	26.10.2020 – 30.10.2020
	Узгодження та затвердження ТЗ	02.11.2020 – 06.11.2020
2. Робочий проект	Розробка та програмування логіки програми	09.11.2020 – 17.11.2020
	Розробка і реалізація інтерфейсу користувача	18.11.2020 – 20.11.2020
	Відлагодження програми	23.11.2020 – 25.11.2020
	Розробка, узгодження та затвердження програмної документації	26.11.2020 – 04.12.2020
3. Впровадження	Підготовка і передача програми та програмної документації замовнику	07.12.2020 – 21.12.2020

7 ПОРЯДОК КОНТРОЛЮ ТА ПРИЙМАННЯ

Контроль здійснюється за допомогою виконання набору тестів з метою знаходження помилок в програмі та його специфікації. Контроль виконання роботи забезпечується керівником розробки.

Прийом програми здійснюється уповноваженою комісією.

БІБЛІОГРАФІЧНИЙ СПИСОК

1. "ДСанПіН 3.3.2-007-98. Державні санітарні правила і норми роботи з візуальними дисплейними терміналами електронно-обчислювальних машин [Текст] / Постанова Головного державного санітарного лікаря України від 10 грудня 1998 р. № 7 – К.,1998.".
2. "ДСН 3.3.6.042-99. Санітарні норми мікроклімату виробничих приміщень [Текст]/ Постанова Головного Державного санітарного лікаря України від 01.12.1999 № 42 - К., 1999.".
3. "ГОСТ 19.101-77. Виды програм и программных документов [Текст]/ Постановление Государственного комитета стандартов Совета Министров СССР от 20 мая 1977 г. – М., 1977.".
4. "Методики оценки трудозатрат по разработке программного обеспечения информационных систем," [Ел. ресурс]. Available: http://repository.enu.kz/bitstream/handle/data/12881/metodika-trudozatrat.pdf . [Дата звернення: 10.12.2020].
5. "Методики оценки трудозатрат," [Ел. ресурс]. Available: http://www.hups.mil.gov.ua/periodic-app/article/11953/soi_2014_8_33.pdf. [Дата звернення: 10.12.2020].
6. "Головне управління статистики у м. Києві," [Ел. ресурс]. Available: http://www.kiev.ukrstat.gov.ua/p.php3?c=1139&lang=1. [Дата звернення: 10.12.2020].
7. "Єдиний соціальний внесок," [Ел. ресурс]. Available: https://index.minfin.com.ua/ua/labour/social/. [Дата звернення: 10.12.2020].
8. "Тарифи на електроенергію," [Ел. ресурс]. Available: https://index.minfin.com.ua/tariff/electric/. [Дата звернення: 10.12.2020].
9. «Інтернет-магазин Comfy» [Ел. ресурс]. Available: https://comfy.ua/noutbuk-igrovoj-hp-pavilion-gaming-15-cx0048-4rn89ea-black.html. [Дата звернення: 10.12.2019].
10. «Системотехнік: середня зарплата в Україні», [Ел. ресурс]. Available: https://www.work.ua/salary-системотехнік. [Дата звернення: 10.12.2019].

11."Сайт роздрібної торгівлі Rozetka," [Ел. ресурс]. Available: https://soft.rozetka.com.ua/microsoft_fqc_09131/p3936301/. [Дата звернення: 10.12.2020].

12. "Сайт роздрібної торгівлі Exe.ua," [Ел. ресурс]. Available: https://exe.ua/en/product/p288620/?utm_medium=cpc&utm_source=hotline&utm_campaign=microsoft-visual-studio-olp&utm_term=microsoft-c5e-01380&utm_content=293377&sku=293377. [Дата звернення: 10.12.2020].

13.«Сайт оголошенн OLX» [Ел. ресурс]. Available: https://www.olx.ua/uk/obyavlenie/sdam-kabinet-16-kv-m-v-ofisom-zdanii-ulblagoeva-31-s-mebelyu-IDJNDsG.html#d542b01665;promoted. [Дата звернення: 10.12.2020].

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

ЗАТВЕРДЖУЮ
Перший проректор Дніпровського
національного університету
залізничного транспорту
імені академіка В. Лазаряна
Б.Є. Боднар

СИСТЕМА ДЛЯ ДОСЛІДЖЕННЯ ЧАСОВОЇ ЕФЕКТИВНОСТІ АЛГОРИТМІВ ВИДАЛЕННЯ НЕВИДИМИХ ЛІНІЙ ТА ПОВЕРХОНЬ

Робочий проект
ЛИСТ ЗАТВЕРДЖЕННЯ
1116130.01193-01-ЛЗ

Завідувач кафедри КІТ
проф. В.І. Шинкаренко

Керівник розробки
доц. О.П. Іванов

Виконавець
студент групи ПЗ1921
І.О. Строчіков

Нормоконтролер
доц. О.С. Куроп'ятник

2020

ЗАТВЕРДЖЕНО
1116130.01193-01-ЛЗ

СИСТЕМА ДЛЯ ДОСЛІДЖЕННЯ ЧАСОВОЇ ЕФЕКТИВНОСТІ АЛГОРИТМІВ ВИДАЛЕННЯ НЕВИДИМИХ ЛІНІЙ ТА ПОВЕРХОНЬ

Специфікація

1116130.01193-01

Аркушів 2

1116130.01193-01

2020

Позначення	Найменування	Примітки
	Документація	
1116130.01193-01-ЛЗ	Лист затвердження	
1116130.01193-01 12 01-ЛЗ	Лист затвердження	
1116130.01193-01 12 01	Текст програми	
1116130.01193-01 13 01-ЛЗ	Лист затвердження	
1116130.01193-01 13 01	Опис програми	
1116130.01193-01 І3 01-ЛЗ	Лист затвердження	
1116130.01193-01 І3 01	Керівництво користувача. Керівництво по роботі з системою дослідження часової ефективності алгоритмів видалення невидимих ліній та поверхонь	

ЗАТВЕРДЖЕНО
1116130.01193-01 13 01-ЛЗ

СИСТЕМА ДЛЯ ДОСЛІДЖЕННЯ ЧАСОВОЇ ЕФЕКТИВНОСТІ АЛГОРИТМІВ ВИДАЛЕННЯ НЕВИДИМИХ ЛІНІЙ ТА ПОВЕРХОНЬ

Опис програми

1116130.01193 13 01

Аркушів 14

2020

АНОТАЦІЯ

Документ 1116130.01193 13 01 «Система для дослідження часової ефективності алгоритмів видалення невидимих ліній та поверхонь. Опис програми» входить до складу програмної документації до дипломного проекту. Даний документ містить опис програми та її функціональної частини.

Програмний продукт реалізований за допомогою мови C# та ігрового движка Unity у середовищі розробки Microsoft Visual Studio 2019.

ЗМІСТ

1 ЗАГАЛЬНІ ВІДОМОСТІ

Програмний продукт «Система для дослідження часової ефективності алгоритмів видалення невидимих ліній та поверхонь» призначений для перевірки та швидкого збору даних щодо ефективності HSR-алгоритмів Z-буферу, scan-line та «художника». Ефективність алгоритмів оцінюється часом роботи відносно одне одного.

На сьогоднішній день існує безліч алгоритмів видалення невидимих ліній та поверхонь, які вирішують задачу видалення тих ділянок зображення, які є прихованими с точки зору глядача у тривимірній комп’ютерній графіці. Але досі не було розроблено такого HSR-алгоритму, який би працював краще інших в абсолютно будь-якому випадку або на будь-якому пристрої. Тому питання найефективнішого використання HSR-алгоритмів у комп’ютерній графіці досі є відкритим.

Основною задачею даного програмного продукту є визначення часу роботи вищезгаданих алгоритмів в залежності від вхідних параметрів та швидкий збір цих даних в зручному вигляді для подальшого аналізу.

Застосування такої програмної системи корисне в тому, щоб збирати дані щодо часу роботи алгоритмів видалення невидимих ліній та поверхонь при відображенні будь-якої тривимірної графіки.

2 ФУНКЦІОНАЛЬНЕ ПРИЗНАЧЕННЯ

За допомогою даного продукту користувач матиме змогу проводити експерименти для оцінки часової ефективності таких алгоритмів видалення невидимих ліній та поверхонь: Z-буферу, scan-line, «художника». Використовуючи різні комбінації вхідних даних, він зможе збирати результати експериментів для подальшого аналізу. Також програма надає можливість зберігати результати експериментів в зручному для аналізу вигляді.

3 ОПИС ЛОГІЧНОЇ СТРУКТУРИ

3.1 Структура програми

Загалом програма має такі складові: логіка HSR-алгоритмів, класи відображення тривимірної моделі, класи проведення тестів, класи взаємодії з файлами.

Схема взаємодії класів зображена на рис. 3.1.

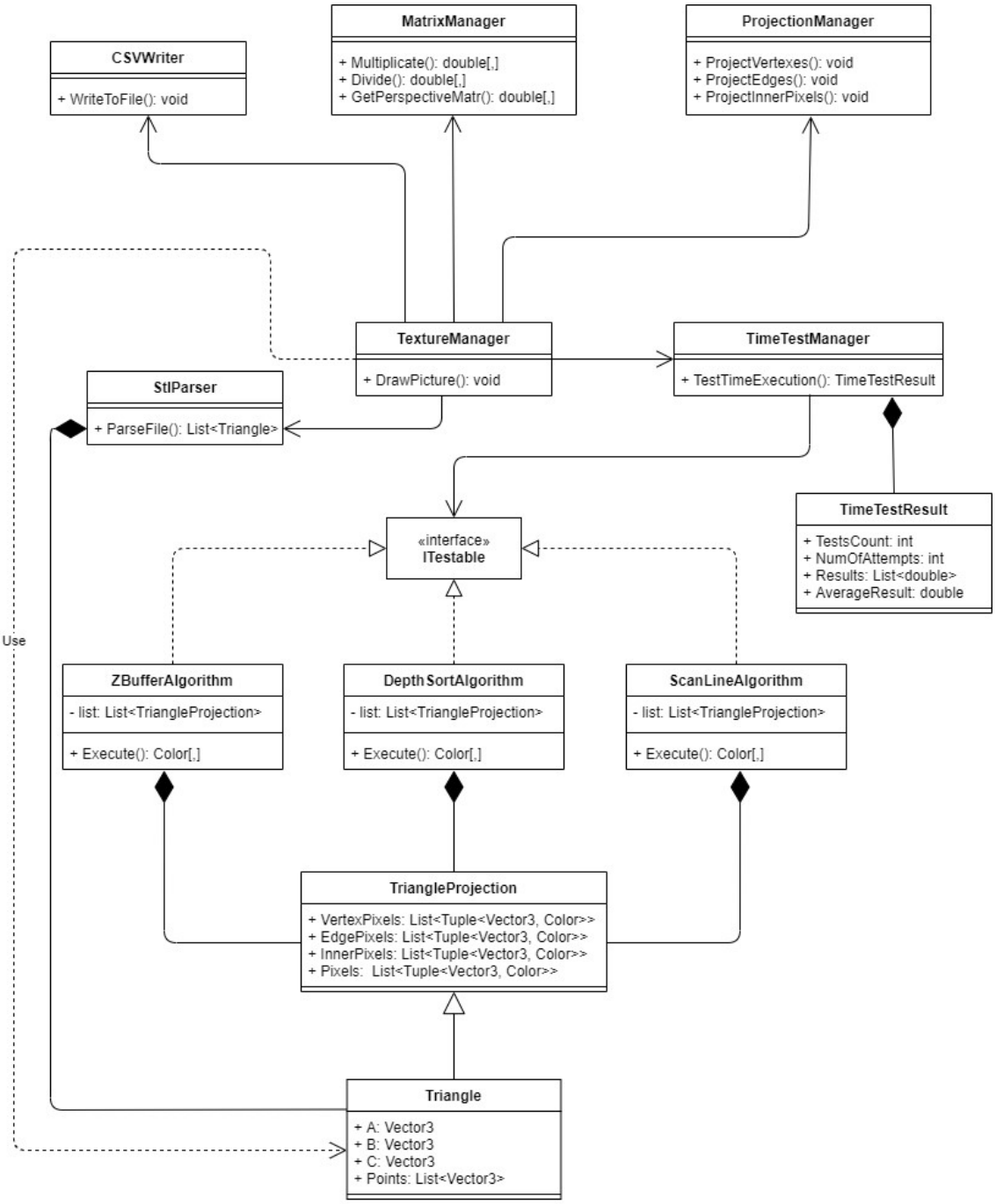

Рисунок 3.1 – Схема взаємодії класів

Детальний опис обов’язків кожного класу:

Клас TextureManager є головним класом у цій системі. Він по черзі викликає методи інших класів, щоб з назви отриманого файлу з моделлю можна було відобразити вже оброблену модель на двовимірному полотні. Одразу він викликає метод ParseFile() класу StlParser та отримує список вершин кожного полігону з координатами у тривимірному просторі. Після цього він викликає методи класу MatrixManager та в результаті отримує матрицю для подальшого проектування. Потім по черзі викликаються методи ProjectVertexes(), ProjectEdges() та ProjectInnerPixels() класу ProjectionManager. Результатом цих викликів є список полігонів, які містять список пікселів вже з координатами для відображення в двовимірному просторі, координатою Z, яка означає дальність від погляду користувача, та кольором пікселя. Після цього викликається метод Execute() одного з обраних HSR-алгоритмів, який повертає двомірну матрицю з кольорами для відображення. Врешті-решт викликаються вбудовані функції Unity, щоб встановити колір пікселю на основі отриманої матриці.

Клас StlParser відповідає за парсинг файлів формату .STL. Слід зауважити, що він може парсити stl-файли лише формату ASCII. Використання бінарних stl-файлів не передбачено системою. Основний метод класу повертає список полігонів з трьома вершинами у тривимірному просторі.

Клас MatrixManager відповідає за арифметичні дії, які пов’язані з матрицями. Крім того, він має метод, який обчислює матрицю проектування по заданим параметрам.

Клас ProjectionManager відповідає за визначення, як саме повинні відображатися отримані полігони на двовимірному полотні. Він має метод ProjectVertexes(), який переводить вершини для відображення у двовимірному просторі. Також він має метод ProjectEdges(), який знаходить пікселі для відображення ребер полігону та отримує їх координати Z. Останній метод ProjectInnerPixels() знаходить всі пікселі для відображення, які знаходяться всередині

полігонів, тобто тут не враховуються лише пікселі, які відносяться до ребер або вершин.

Клас Triangle – це клас-модель, який представляє собою полігон з трьома вершинами у тривимірному просторі. Ця модель використовується класом StlParser, коли він зберігає координати точок з файлу у цій моделі.

Клас TrianlgeProjection – це теж клас-модель, який є спадкоємцем класу Triangle. Це вже модель для представлення полігону у двовимірному просторі. Він використовується класом ProjectionManager та усіма класами, які реалізовують деякий HSR-алгоритм.

Інтерфейс ITestable – це інтерфейс, який реалізовується класами HSR-алгоритмів. Він потрібен для того, щоб можна було абстрагуватися від конкретної реалізації HSR-алгоритму. Зокрема, цей інтерфейс потрібен класу TimeTestManager, який викликає метод Execute() класу, який реалізує цей інтерфейс в даний момент.

Клас ZBufferAlgorithm – це клас, який реалізує інтерфейс ITestable та містить у собі метод Execute з логікою видалення невидимих ліній та поверхонь за допомогою алгоритму Z-буфера.

Клас ScanLineAlgorithm – це клас, який реалізує інтерфейс ITestable та містить у собі метод Execute з логікою видалення невидимих ліній та поверхонь за допомогою scan-line алгоритму.

Клас DepthSortAlgorithm – це клас, який реалізує інтерфейс ITestable та містить у собі метод Execute з логікою видалення невидимих ліній та поверхонь за допомогою алгоритму «художника».

Клас TimeTestManager відповідає за тестування обраного HSR-алгоритму. Його основний метод TestTimeExecution() приймає об’єкт інтерфейсу, який містить у собі об’єкт конкретної реалізації HSR-алгоритму. Цей метод запускає метод Execute() задану кількість разів та повертає об’єкт класу TimeTestResult.

Клас TimeTestResult – це клас-модель, який представляє собою отримані результати експерименту. Він містить у собі необхідні поля для подальшого запису у файл результатів.

Клас CSVWriter відповідає за запис результатів експерименту у визначений файл формату .CSV.

3.2 Алгоритм програми

Програма має наступний алгоритм роботи:

1) Очікування вводу потрібних параметрів експерименту, в тому числі й вибору файлу з тривимірною моделлю та HSR-алгоритму для обробки.
2) Парсинг файлу з тривимірною моделлю.
3) Запуск обробки моделі HSR-алгоритмом обрану кількість разів та фіксування часу роботи.
4) Збереження часу роботи алгоритму у файлі.
5) Вивід візуального результату обробки (тривимірна модель з видаленими невидимими лініями) на екран.

3.3 Використані методи

Часова ефективність кожного з досліджуваних алгоритмів обчислюється за результатами експериментів. Дуже важливо виконати експеримент декілька сотень разів, щоб більш точно встановити середній показник часу. Середній показник часу для одного експерименту буде обчислюватися за такою формулою:

$$T_{avg} = \frac{\sum_{i=0}^{n} t_i}{n}, \tag{3.1}$$

де T_{avg} – середній час роботи алгоритму при заданих вхідних даних,

n – кількість експериментів,

t_i – час роботи алгоритму в і-тому експерименті.

3.4 Зв’язки програми з іншими програмами

Даний програмний продукт є повністю самостійним та не потребує використання іншого програмного забезпечення.

4 ВИКЛИК І ЗАВАНТАЖЕННЯ

Перед запуском програми необхідно переконатися, що папка з даними, необхідними для роботи додатку, присутня та не була пошкоджена або видалена. Запуск програми є простим та не потребує якихось спеціальних дій. Для запуску необхідно просто двічі натиснути лівою кнопкою миші по іконці програми.

5 ВИКОРИСТАНІ ТЕХНІЧНІ ЗАСОБИ

Для коректної роботи програма повинна виконуватися на Windows-сумісних комп’ютерах, що мають такі мінімальні характеристики:

– 2-ядерний процесор з тактовою частотою не менше 1,6 ГГц;

– 8 ГБ доступної оперативної пам’яті;

– жорсткий диск на 128 ГБ;

– пристрій виводу зображення з роздільною здатністю екрану 1920x1080.

Вимога до інформаційної і програмної сумісності лише одна – операційна система Windows 10 має бути версії 20H2 і пізніше.

6 ОПИС ПРИЗНАЧЕНОГО ДЛЯ КОРИСТУВАЧА ІНТЕРФЕЙСУ

Після запуску користувач одразу побачить екран налаштування експерименту (рис. 3.2). Екран налаштування візуально розбитий на чотири секції, кожна з яких має своє призначення:

- секція Resolution;
- секція HSR Algorithm;
- секція Testing;
- секція Choose file.

Секція Resolution містить два поля для вводу. Перше поле відповідає за ширину зображення, а друге – за висоту. Ці поля мають текст Width та Height відповідно, який замінюється числами, які користувач може ввести. Це було зроблено для того, щоб користувач інтуїтивно розумів, яке поле відповідає за ширину, а яке – за висоту та випадково їх не сплутав.

Секція HSR Algorithm містить три перемикача (radio button). В залежності від обраного алгоритму, користувач може поставити флаг для одного з них. Слід замітити, що користувач може обрати лише один HSR-алгоритм. Якщо він спробує встановити флаг над іншим алгоритмом, то попередній флаг буде знято. Це було зроблено, щоб запобігти неоднозначності.

Секція Testing містить лише одне поле для вводу. Це поле відповідає за те, скільки разів буде запущено HSR-алгоритм. Користувач може ввести будь-яке число більше 0.

Остання секція Choose file містить лише одне спадне меню (dropdown). В закритому стані воно відображає файл, який на даний момент обрано для експерименту. Користувач може натиснути на нього, після чого меню розкриється та відобразить усі варіанти для вибору. Після цього користувач може обрати необхідну модель зі списку та натиснути на обраний варіант, після чого меню знову закриється, а відображатися буде вже щойно обраний варіант.

Екран налаштування має велику кнопку «Apply» знизу, натискання на яку запустить рендеринг обраної моделі на полотні обраного розміру у секції Resolution. Після цього екран налаштування зникне і з’явиться екран з відображеною тривимірною моделлю (рис. 3.3).

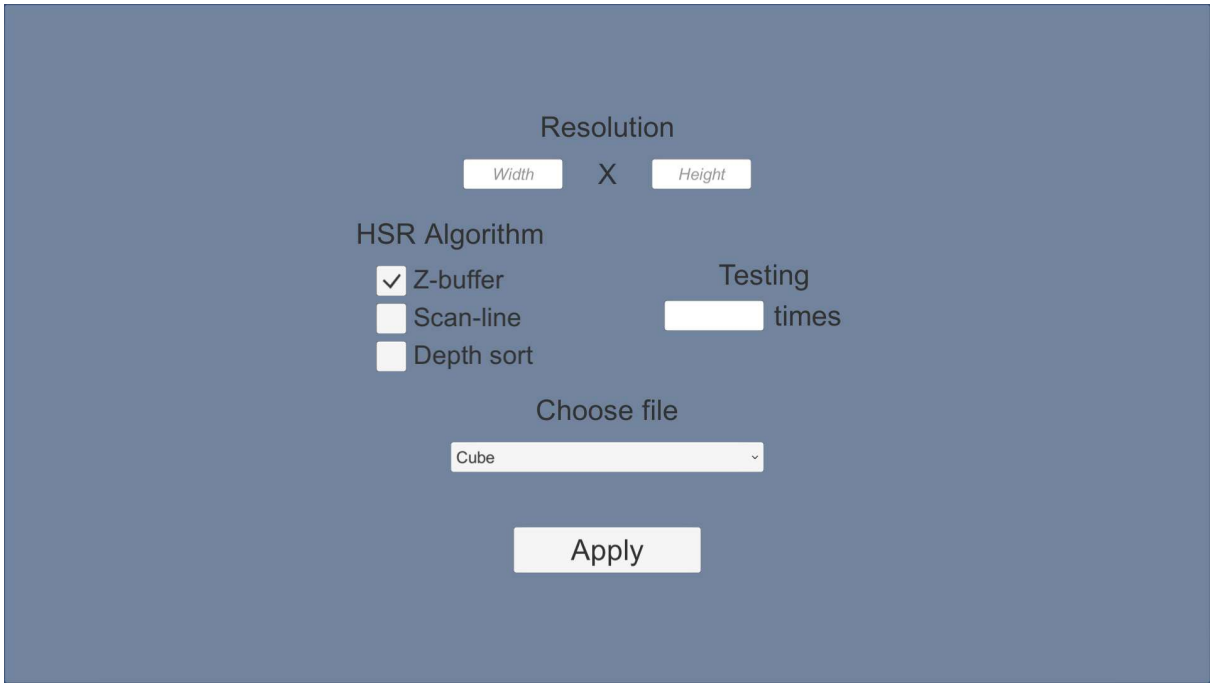

Рисунок 3.2 – Екран налаштування експерименту

Екран з відображеною тривимірною моделлю має лише два елементи. Перший елемент – це полотно (canvas). На цьому полотні відображається обрана раніше статична тривимірна модель. Другий елемент – це кнопка з назвою «Return». Натискання на цю кнопку повертає користувача до екрану налаштування експерименту. Після цього екран з відображеною тривимірною моделлю зникає.

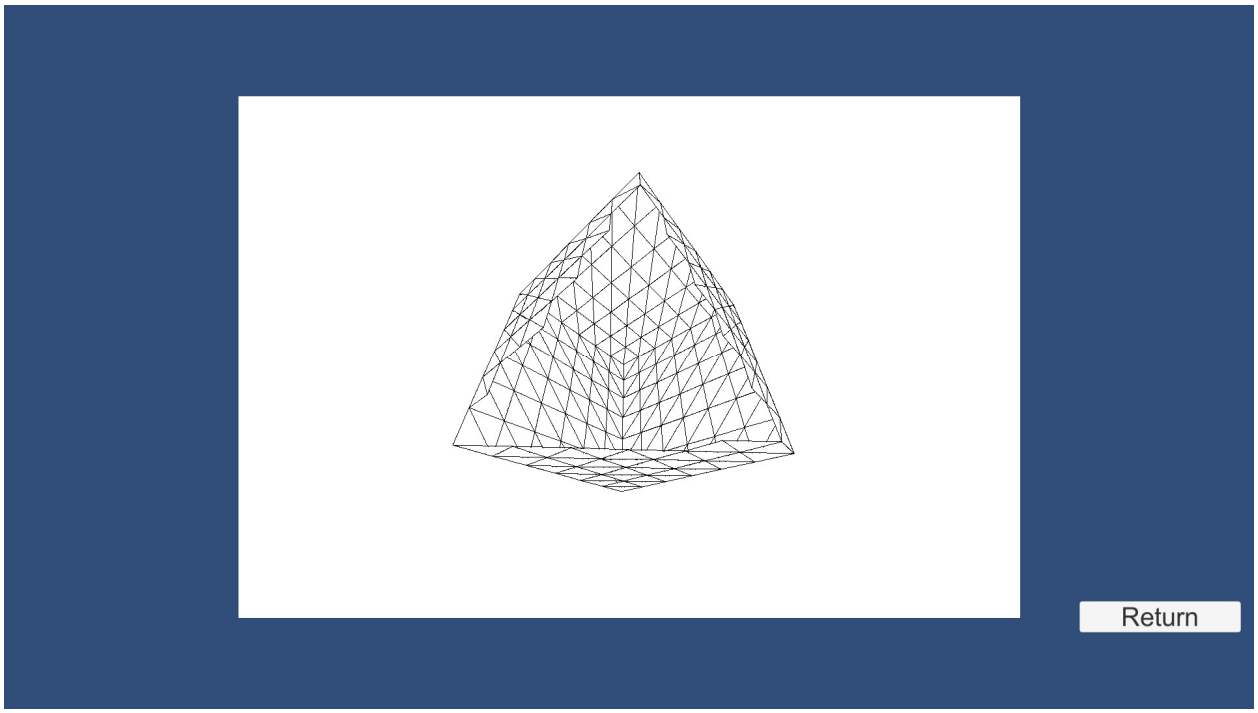

Рисунок 3.3 – Екран з відображеною тривимірною моделлю

ЗАТВЕРДЖЕНО
1116130.01193-01 І3 01-ЛЗ

СИСТЕМА ДЛЯ ДОСЛІДЖЕННЯ ЧАСОВОЇ ЕФЕКТИВНОСТІ АЛГОРИТМІВ ВИДАЛЕННЯ НЕВИДИМИХ ЛІНІЙ ТА ПОВЕРХОНЬ

Керівництво користувача. Керівництво по роботі з системою для дослідження часової ефективності алгоритмів видалення невидимих ліній та поверхонь

1116130.01193-01 І3 01

Аркушів 8

2020

АНОТАЦІЯ

Документ 1116130.01193-01 І3 01 «Система для дослідження часової ефективності алгоритмів видалення невидимих ліній та поверхонь. Керівництво користувача. Керівництво по роботі з системою для дослідження часової ефективності алгоритмів видалення невидимих ліній та поверхонь» входить до складу програмної документації до дипломного проекту. Даний документ містить опис призначення та умов застосування програми, опис операцій.

ЗМІСТ

ВСТУП

Програмний продукт «Система для дослідження часової ефективності алгоритмів видалення невидимих ліній та поверхонь» призначений для перевірки та швидкого збору даних щодо ефективності HSR-алгоритмів Z-буферу, scan-line та «художника». Ефективність алгоритмів оцінюється часом роботи відносно одне одного.

На сьогоднішній день існує безліч алгоритмів видалення невидимих ліній та поверхонь, які вирішують задачу видалення тих ділянок зображення, які є прихованими с точки зору глядача у тривимірній комп'ютерній графіці. Але досі не було розроблено такого HSR-алгоритму, який би працював краще інших в абсолютно будь-якому випадку або на будь-якому пристрої. Тому питання найефективнішого використання HSR-алгоритмів у комп'ютерній графіці досі є відкритим.

Основною задачею даного програмного продукту є визначення часу роботи вищезгаданих алгоритмів в залежності від вхідних параметрів та швидкий збір цих даних в зручному вигляді для подальшого аналізу.

Застосування такої програмної системи корисне в тому, щоб збирати дані щодо часу роботи алгоритмів видалення невидимих ліній та поверхонь при відображенні будь-якої тривимірної графіки.

1 ПРИЗНАЧЕННЯ ТА УМОВИ ЗАСТОСУВАННЯ

1.1 Функціонал програми

Програма повинна:

- мати можливість працювати з файлами формату .STL;
- надавати можливість ввести параметри експерименту (роздільна здатність, кількість тестів, тривимірна модель, HSR-алгоритм);
- обробляти тривимірну модель обраним HSR-алгоритмом.
- зберігати часові результати експерименту у файл;
- відображати оброблену тривимірну модель.

Вхідні дані:

- ширина та висота зображення (цілі числа);
- кількість тестів (ціле число);
- унікальний номер HSR-алгоритму, який буде використано (ціле число);
- файл формату .STL, який потрібно обробити (текст).

Вихідні дані:

- візуальне відображення обраної тривимірної моделі;
- файл формату .CSV, який містить результати експерименту та має такі дані:
    a. використаний HSR-алгоритм;
    b. роздільна здатність екрану;
    c. кількість полігонів моделі;
    d. час роботи алгоритму.

1.2 Вимоги до складу і параметрів технічних засобів

Для коректної роботи програма повинна виконуватися на Windows-сумісних комп’ютерах, що мають такі мінімальні характеристики:

– 2-ядерний процесор з тактовою частотою не менше 1,6 ГГц;
– 8 ГБ доступної оперативної пам’яті;
– жорсткий диск на 128 ГБ;
– пристрій виводу зображення з роздільною здатністю екрану 1920x1080.

1.3 Вимоги до інформаційної і програмної сумісності

Вимога до інформаційної і програмної сумісності лише одна – операційна система Windows 10 має бути версії 20H2 і пізніше.

2 ПІДГОТОВКА ДО РОБОТИ

Для роботи з даним програмним продуктом користувачу необхідно мати:

- персональний комп’ютер;
- встановлену операційну систему Windows 10;
- девайси для взаємодії з персональним комп’ютером – клавіатуру, комп’ютерну мишу та екран.

2.1 Запуск програмного додатку

Для запуску програмного додатку потрібно виконати наступні команди у визначеному порядку:

1) перейти до папки, яка містить виконуваний файл з назвою SurfaceRemovalProject.exe;
2) переконатися, що ця папка також містить папку с даними з назвою SurfaceRemovalProject_Data;
3) двічі натиснути лівою кнопкою миші по іконці виконуваному файлу.

3 ОПИС ОПЕРАЦІЙ

Запустіть програмний додаток та перейдіть до екрану налаштування експерименту. На рис. 3.1 представлений вигляд екрану налаштування експерименту. В полі «Width» та «Height» введіть бажану ширину та висоту зображення. Потім оберіть один з HSR-алгоритмів для обробки. Після цього введіть бажану кількість тестів, які потрібно провести у цьому випробуванні. Потім оберіть одну із запропонованих тривимірних моделей зі списку. Переконайтеся, що всі поля заповнені та натискайте на кнопку «Apply».

Після натискання на кнопку, ви опинитеся на екрані з відображеною тривимірною моделлю. На рис. 3.2 зображено екран з відображеною тривимірною моделлю. Після цього натисніть на кнопку «Return» або закрийте програму. Якщо вам потрібно повернутися до екрану налаштування експерименту, натисніть на кнопку «Return». Якщо вам потрібно ознайомитися з отриманими результатами, закрийте програму та перейдіть до папки SurfaceRemovalProject_Data, яка знаходиться поряд з виконуваним файлом. Потім відкрийте файл Result.csv та ознайомтеся з результатами експерименту.

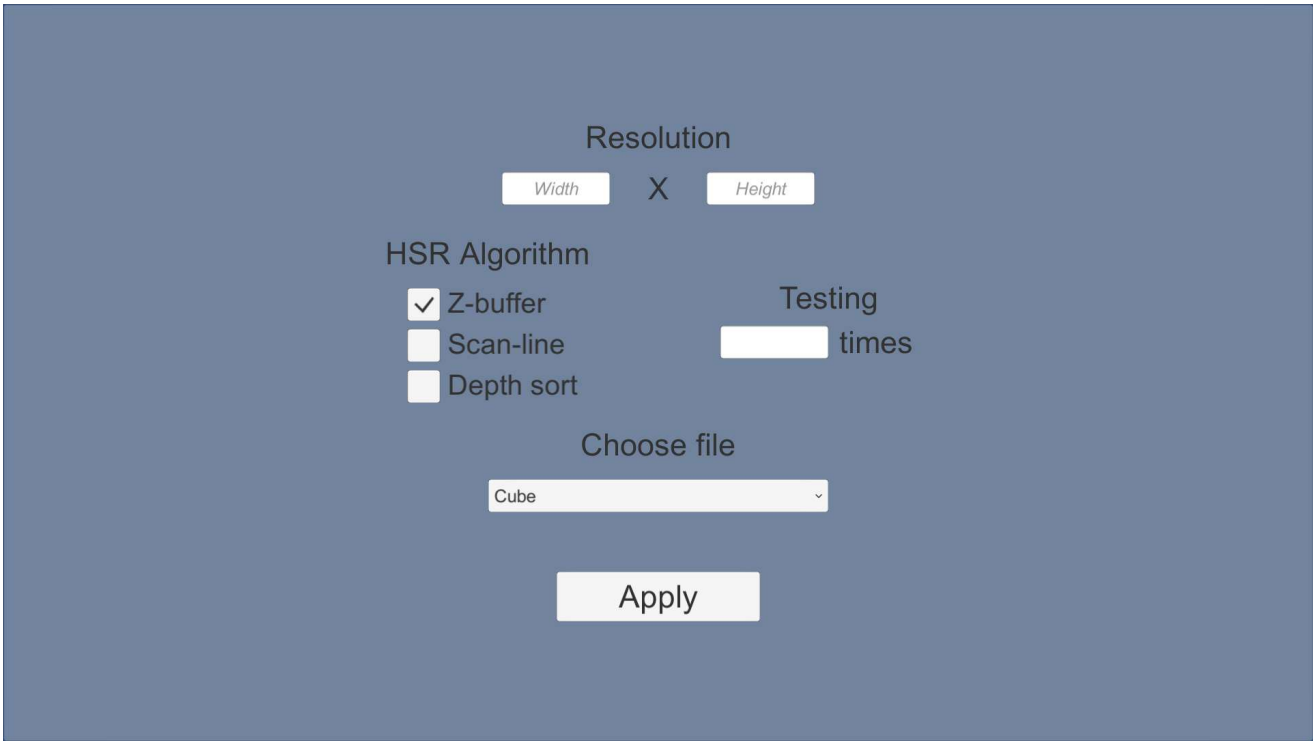

Рисунок 3.1 – Екран налаштування експерименту

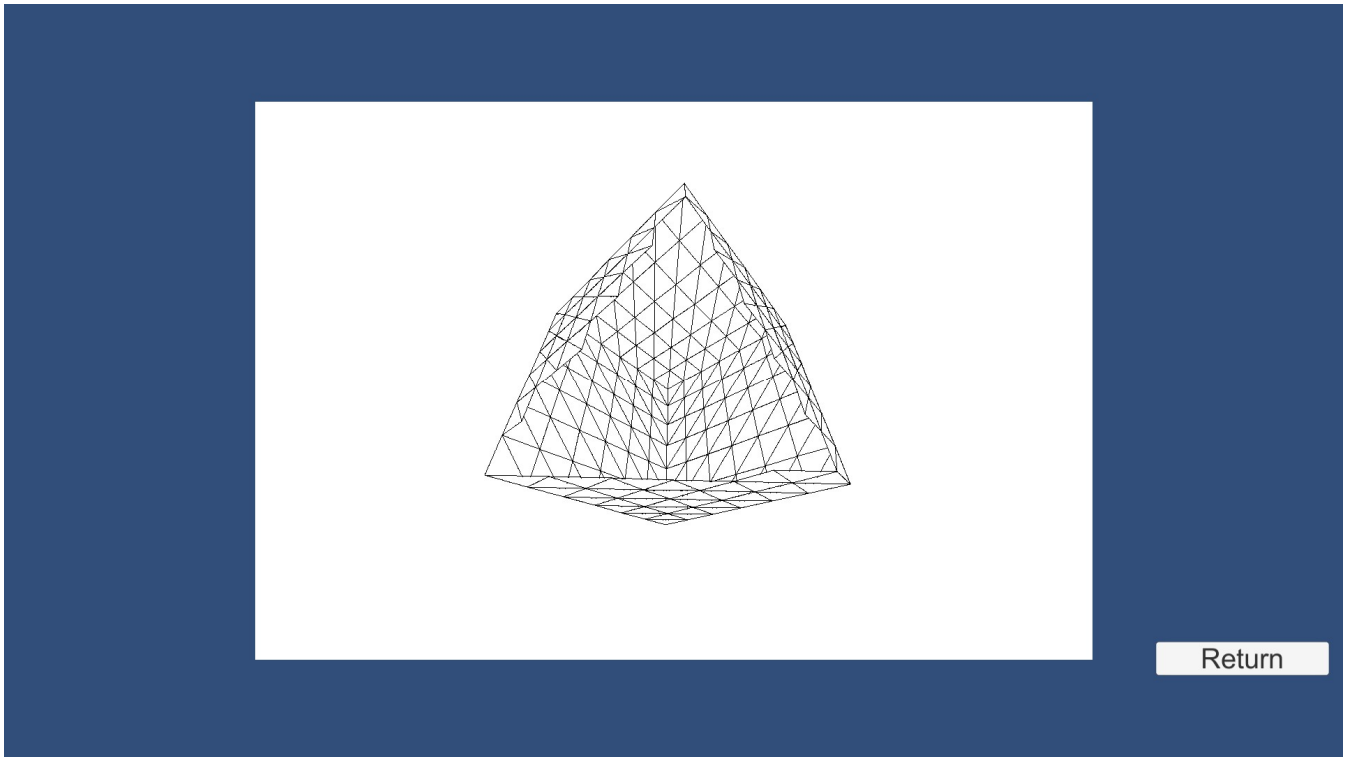

Рисунок 3.2 – Екран з відображеною тривимірною моделлю

ЗАТВЕРДЖЕНО
1116130.01193-01 12 01-ЛЗ

СИСТЕМА ДЛЯ ДОСЛІДЖЕННЯ ЧАСОВОЇ ЕФЕКТИВНОСТІ АЛГОРИТМІВ ВИДАЛЕННЯ НЕВИДИМИХ ЛІНІЙ ТА ПОВЕРХОНЬ

Текст програми

1116130.01193-01 12 01

Аркушів 14

2020

АНОТАЦІЯ

Документ 1116130.01193-01 12 01 «Система для дослідження часової ефективності алгоритмів видалення невидимих ліній та поверхонь. Текст програми» входить до складу програмної документації до дипломного проекту. Даний документ містить структуру програми та текст програмного коду.

ЗМІСТ

1 СТРУКТУРА ПРОГРАМИ

Структура програми зображена на рис 1.1.

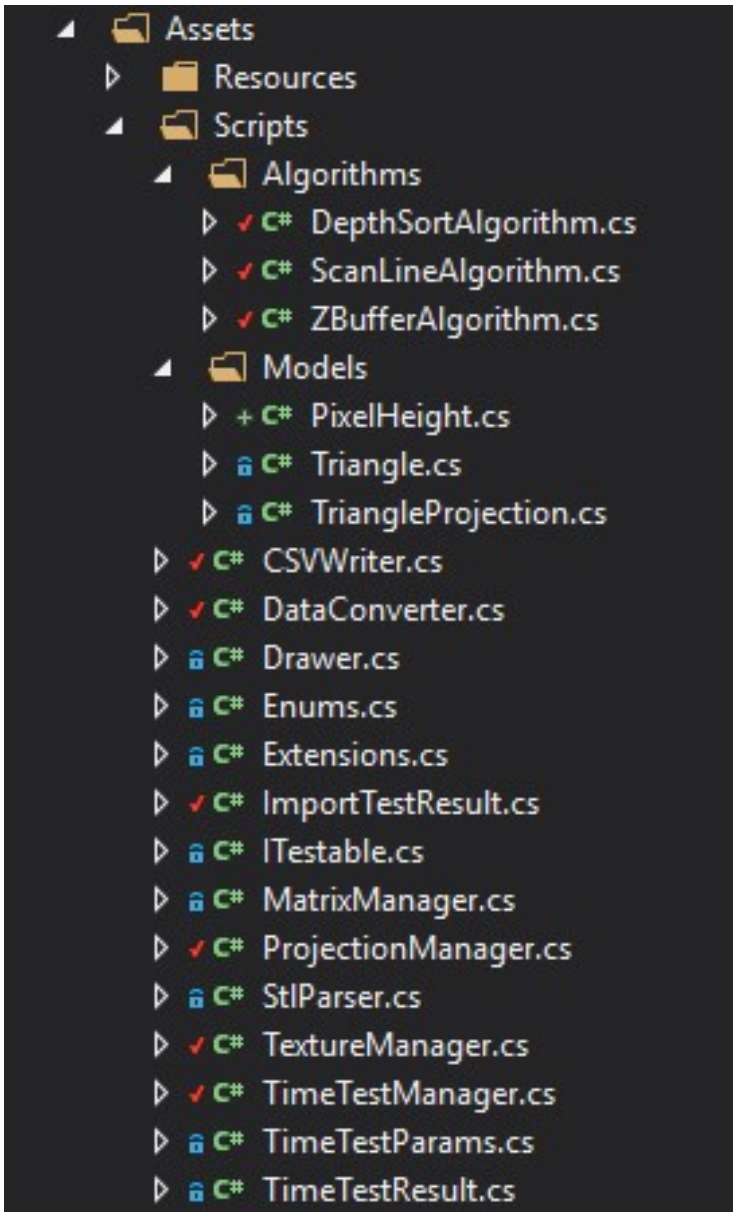

Рисунок 1.1 – Структура програми

2 ТЕКСТ ПРОГРАМИ

TextureManager.cs

```
using Assets.Scripts;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.UI;
using Assets.Scripts.Models;
using Assets.Scripts.Algorithms;
using System.Linq;

public class TextureManager : MonoBehaviour
{
    public Dropdown dropdown;
    public Image image;
    private Texture2D texture;

    public GameObject goImage;
    public GameObject goParams;

    private int width;
    private int height;

    private int testsCount;
    private int delay = 0;

    private List<Triangle> vertexesList;
    double[,] perspectiveMatr;
    Enums.Algorithm algorithm =
Enums.Algorithm.ZBuffer;
    TextAsset[] files;

    // Start is called before the first frame
update
    void Start()
    {
        dropdown.ClearOptions();
        files =
Resources.LoadAll<TextAsset>("Models");

        dropdown.AddOptions(files.Select(x =>
x.name).ToList());
    }

    private void DrawPicture()
    {
        texture = new Texture2D(width,
height);

        List<TriangleProjection>
triangleProjections = new
List<TriangleProjection>();
        int count = 0;
        foreach (var triangle in
vertexesList)
        {
            List<Vector3> projPoints = new
List<Vector3>();
            bool visible = true;
            foreach (var point in
triangle.Points)
            {
                var result =
MatrixManager.Multiplicate(new double[,] { {
point.x, point.y, point.z, 1 } },
perspectiveMatr);
                result =
MatrixManager.Divide(result, result[0, 3]);
                if (!(result[0, 0] > -1 &&
result[0, 0] < 1 && result[0, 1] > -1 &&
result[0, 1] < 1 && result[0, 2] > 0 &&
result[0, 2] < 1))
                {
                    visible = false;
                }
                projPoints.Add(new Vector3 {
x = (float)result[0, 0], y = (float)result[0,
1], z = (float)result[0, 2] });
            }

            if (visible)
                triangleProjections.Add(new
TriangleProjection { Points = projPoints });
        }

        foreach (var item in
triangleProjections)
        {

ProjectionManager.ProjectVertexes(item,
width, height);

ProjectionManager.ProjectEdges(item);

ProjectionManager.ProjectInnerPoints(item);
        }

        ITestable method = new
ZBufferAlgorithm(triangleProjections, width,
height);
        switch (algorithm)
        {
            case Enums.Algorithm.ZBuffer:
                method = new
ZBufferAlgorithm(triangleProjections, width,
height);
                break;
            case Enums.Algorithm.ScanLine:
                method = new
ScanLineAlgorithm(triangleProjections, width,
height);
                break;
            case Enums.Algorithm.DepthSort:
                method = new
DepthSortAlgorithm(triangleProjections,
width, height);
                break;
        }
```

```
        Color[,] toDraw = method.Execute();

        for (int i = 0; i < width; i++)
            for (int j = 0; j < height; j++)
            {
                if (toDraw[i, j] !=
Color.clear)
                    texture.SetPixel(i, j,
toDraw[i, j]);
                else
                    texture.SetPixel(i, j,
Color.white);
            }

        texture.Apply();

        Sprite s = Sprite.Create(texture, new
Rect(0, 0, width, height), Vector2.zero);
        image.sprite = s;

        TimeTestParams timeTestParams = new
TimeTestParams(1, testsCount, delay);
        var testResult =
TimeTestManager.TestTimeExecution(method,
timeTestParams);

        List<ImportTestResult>
importTestResults = new
List<ImportTestResult>();
        foreach (var time in
testResult.Results)
        {
            importTestResults.Add(new
ImportTestResult
            {
                AlgorithmName =
algorithm.ToString(),
                Width = width,
                Height = height,
                VerticesCount =
triangleProjections.Count(),
                PixelsCount =
triangleProjections.Sum(x => x.Pixels.Count),
                Result = time
            });
        }


CSVWriter.WriteToFile(importTestResults);
    }

    public void ChangeAlgorithm(int
algorithmNum)
    {
        Enums.Algorithm choosenAlgorithm =
(Enums.Algorithm)algorithmNum;

        switch (choosenAlgorithm)
        {
            case Enums.Algorithm.ZBuffer:
                if
(GameObject.Find("ZBufferToggle").GetComponen
t<Toggle>().isOn)
                {

GameObject.Find("ScanlineToggle").GetComponen
t<Toggle>().isOn = false;

GameObject.Find("DepthSortToggle").GetCompone
nt<Toggle>().isOn = false;
                    algorithm =
choosenAlgorithm;
                }
                break;
            case Enums.Algorithm.ScanLine:
                if
(GameObject.Find("ScanlineToggle").GetCompone
nt<Toggle>().isOn)
                {

GameObject.Find("ZBufferToggle").GetComponent
<Toggle>().isOn = false;

GameObject.Find("DepthSortToggle").GetCompone
nt<Toggle>().isOn = false;
                    algorithm =
choosenAlgorithm;
                }
                break;
            case Enums.Algorithm.DepthSort:
                if
(GameObject.Find("DepthSortToggle").GetCompon
ent<Toggle>().isOn)
                {

GameObject.Find("ZBufferToggle").GetComponent
<Toggle>().isOn = false;

GameObject.Find("ScanlineToggle").GetComponen
t<Toggle>().isOn = false;
                    algorithm =
choosenAlgorithm;
                }
                break;
        }
    }

    public void ApplyChanges()
    {
        width =
GameObject.Find("WidthRes").GetComponent<Inpu
tField>().text.ToInt();
        height =
GameObject.Find("HeightRes").GetComponent<Inp
utField>().text.ToInt();

        testsCount =
GameObject.Find("TestsCount").GetComponent<In
putField>().text.ToInt();

        string textToParse =
files[dropdown.value].text;
        vertexesList =
StlParser.ParseFile(textToParse);

        perspectiveMatr =
MatrixManager.GetPerspectiveMatrix(5, 1000,
90, new Vector2Int(width, height));
```

```
        goImage.SetActive(true);
        goParams.SetActive(false);

goImage.transform.Find("Image").localScale =
new Vector3(width, height);

        DrawPicture();
    }
```

StlParser.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using UnityEngine;
using Assets.Scripts.Models;
using System.IO;
using System.Text.RegularExpressions;
using System.Globalization;

namespace Assets.Scripts
{
    public static class StlParser
    {
        public static List<Triangle>
ParseFile(string text)
        {
            List<Triangle> facets = new
List<Triangle>();
            const string regexSolid =
@"solid\s+(?<Name>[^\r\n]+)?";

            var textWithoutHeaders =
text.Replace("\r", "").Split('\n');
            textWithoutHeaders =
textWithoutHeaders.Where(x => !Regex.Match(x,
regexSolid).Success).ToArray();
            text = string.Join("\r\n",
textWithoutHeaders);

            using (StringReader sr = new
StringReader(text))
            {
                Triangle currentFacet = null;

                //Read each facet until the
end of the stream.
                while ((currentFacet =
ReadNextFacet(sr)) != null)
                {
                    facets.Add(currentFacet);
                }

            }
            return facets;
        }

        public static MemoryStream
GenerateStreamFromString(string s)
        {
            var stream = new MemoryStream();
            var writer = new
StreamWriter(stream);
            writer.Write(s);
            writer.Flush();
            stream.Position = 0;
            return stream;
        }

        private static Triangle
ReadNextFacet(StringReader sr)
        {
            if (sr == null)
                return null;

            //Create the facet.
            Triangle facet = new Triangle();

            //Read the "normal".
            sr.ReadLine();

            //Skip the "outer loop".
            sr.ReadLine();

            //Read 3 vertices.
            facet.A = ReadNextVertex(sr);
            facet.B = ReadNextVertex(sr);
            facet.C = ReadNextVertex(sr);

            //Read the "endloop" and
"endfacet".
            sr.ReadLine();
            sr.ReadLine();

            if (facet.A == null || facet.B ==
null || facet.C == null)
                return null;

            return facet;
        }

        private static Vector3?
ReadNextVertex(StringReader sr)
        {
            const string regex =
@"\s*(vertex)\s+(?<X>[^\s]+)\s+(?<Y>[^\s]+)\s
+(?<Z>[^\s]+)";
            const NumberStyles numberStyle =
(NumberStyles.AllowExponent |
NumberStyles.AllowDecimalPoint |
NumberStyles.AllowLeadingSign);

            float x, y, z;

            //Read the next line of data.
            string data = sr.ReadLine();
    public void ReturnToParams()
    {
        goImage.SetActive(false);
        goParams.SetActive(true);
    }
}
```

```
            if (data == null)
                return null;

            //Ensure that the data is
formatted correctly.
            Match match = Regex.Match(data,
regex, RegexOptions.IgnoreCase);

            if (!match.Success)
                return null;

            //Parse the three coordinates.
            if
(!float.TryParse(match.Groups["X"].Value,
numberStyle, CultureInfo.InvariantCulture,
out x))
                throw new
FormatException("Could not parse X coordinate
\"{0}\" as a decimal.");
```

TimeTestManager.cs

```
using System;
using System.Collections.Generic;
using System.Diagnostics;
using System.Linq;
using System.Text;
using System.Threading;
using System.Threading.Tasks;

namespace Assets.Scripts
{
    public static class TimeTestManager
    {
        public static TimeTestResult
TestTimeExecution(ITestable algorithm,
TimeTestParams testParams)
        {
            TimeTestResult timeTestObject =
new TimeTestResult
            {
                NumOfAttempts =
testParams.NumOfAttempts,
                TestsCount =
testParams.TestsCount,
                Results = new List<double>()
            };


            for (int attemt = 0; attemt <
testParams.NumOfAttempts; attemt++)
```

CSVWriter.cs

```
using System.Collections.Generic;
using System.IO;
using UnityEngine;

namespace Assets.Scripts
{
    public static class CSVWriter
    {
            if
(!float.TryParse(match.Groups["Y"].Value,
numberStyle, CultureInfo.InvariantCulture,
out y))
                throw new
FormatException("Could not parse Y coordinate
\"{0}\" as a decimal.");

            if
(!float.TryParse(match.Groups["Z"].Value,
numberStyle, CultureInfo.InvariantCulture,
out z))
                throw new
FormatException("Could not parse Z coordinate
\"{0}\" as a decimal.");

            return new Vector3(x, y, z);
        }
    }
}
```

```
            {
                GC.Collect();
                Stopwatch stopWatch = new
Stopwatch();
                stopWatch.Start();

                for (int i = 0; i <
testParams.TestsCount; i++)
                {
                    algorithm.Execute();
                }

                stopWatch.Stop();
                TimeSpan ts =
stopWatch.Elapsed;
                double elapsedTime =
ts.Minutes * 60000 + ts.Seconds * 1000 +
ts.Milliseconds;


timeTestObject.Results.Add(elapsedTime /
testParams.TestsCount);

Thread.Sleep(testParams.Interval);
            }

            return timeTestObject;
        }
    }
}
```

```
        private static string Path =
Application.dataPath;
        public static void
WriteToFile(List<ImportTestResult>
resultsToSave)
        {
            List<string> headers = new
List<string> { "Algorithm name", "Width",
"Height", "Polygons count", "Pixels count",
"Result (ms)" };
            bool writeHeaders;
```

```
            writeHeaders = !File.Exists(Path
+ "\\Results.csv");

            using (StreamWriter sw = new
StreamWriter(Path + "\\Results.csv", true))
            {
                if (writeHeaders)
                {
                    for (int i = 0; i <
headers.Count; i++)
                    {
                        sw.Write(headers[i]);
                        if (i + 1 !=
headers.Count)
                            sw.Write(",");
                    }
                    sw.WriteLine();
```

ITestable.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using UnityEngine;
```

ZBufferAlgorithm,cs

```
using Assets.Scripts.Models;
using System;
using System.Collections.Generic;
using System.Diagnostics;
using System.Linq;
using System.Text;
using System.Threading;
using System.Threading.Tasks;
using UnityEngine;

namespace Assets.Scripts.Algorithms
{
    public class ZBufferAlgorithm : ITestable
    {
        private List<TriangleProjection>
_projections;
        private int _width;
        private int _height;

        public
ZBufferAlgorithm(List<TriangleProjection>
projections, int width, int height)
        {
            _projections = projections;
            _width = width;
            _height = height;
        }

        public Color[,] Execute()
        {
            Color[,] colorBuffer = new
Color[_width, _height];
            float[,] zBuffer = new
float[_width, _height];
                }

                for (int i = 0; i <
resultsToSave.Count; i++)
                {

sw.WriteLine($"{resultsToSave[i].AlgorithmNam
e}, {resultsToSave[i].Width},
{resultsToSave[i].Height},
{resultsToSave[i].VerticesCount},
{resultsToSave[i].PixelsCount},
{resultsToSave[i].Result}");
                }
            }
        }
    }
}
```

```
namespace Assets.Scripts
{
    public interface ITestable
    {
        Color[,] Execute();
    }
}
```

```
            for (int i = 0; i < _width; i++)
                for (int j = 0; j < _height;
j++)
                {
                    zBuffer[i, j] = 1;
                }

            foreach (var polygon in
_projections)
            {
                foreach (var point in
polygon.Pixels)
                {
                    int x =
(int)point.Item1.x;
                    int y =
(int)point.Item1.y;

                    if (point.Item1.z <
zBuffer[x, y])
                    {
                        zBuffer[x, y] =
point.Item1.z;
                        colorBuffer[x, y] =
point.Item2;
                    }
                }
            }

            return colorBuffer;
        }
    }
}
```

ScanLineAlgorithm.cs

```
using Assets.Scripts.Models;
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using UnityEngine;

namespace Assets.Scripts.Algorithms
{
    public class ScanLineAlgorithm :
ITestable
    {
        private Dictionary<int,
Dictionary<int, List<Tuple<float, Color>>>>
_dict;
        private List<PixelHeight>[,] _array;
        private int _width;
        private int _height;

        public
ScanLineAlgorithm(List<TriangleProjection>
projections, int width, int height)
        {
            _array =
DataConverter.ConvertToArray(projections,
width, height);
            _width = width;
            _height = height;
        }

        public Color[,] Execute()
        {
            Color[,] colorBuffer = new
Color[_width, _height];

            for (int i = 0; i < _width; i++)
                for (int j = 0; j < _height;
j++)
                {
                    if (_array[i, j] != null)
                    {
                        var points =
_array[i, j];

                        if (points.Count ==
0)
                        {
                            colorBuffer[i, j]
= Color.white;
                        }
                        else if (points.Count
== 1)
                        {
                            colorBuffer[i, j]
= points[0].Color;
                        }
                        else
                        {
                            float minZ =
points[0].Z;
                            Color colorMinZ =
points[0].Color;
                            for (int x = 1; x
< points.Count; x++)
                            {
                                if
(points[x].Z < minZ)
                                {
                                    minZ =
points[x].Z;
                                    colorMinZ
= points[x].Color;
                                }
                            }
                            colorBuffer[i, j]
= colorMinZ;
                        }
                    }
                }

            return colorBuffer;
        }
    }
}
```

DepthSortAlgorithm.cs

```
using Assets.Scripts.Models;
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using UnityEngine;

namespace Assets.Scripts.Algorithms
{
    public class DepthSortAlgorithm :
ITestable
    {
        private List<TriangleProjection>
_projections;
        private int _width;
        private int _height;

        public
DepthSortAlgorithm(List<TriangleProjection>
projections, int width, int height)
        {
            _projections = projections;
            _width = width;
            _height = height;
        }

        public Color[,] Execute()
        {
            Color[,] colorBuffer = new
Color[_width, _height];
            _projections.Sort(delegate
(TriangleProjection p1, TriangleProjection
p2)
            {
```

```
                float avgZ1 =
p1.VertexPixels.Sum(x => x.Item1.z) /
p1.VertexPixels.Count;
                float avgZ2 =
p2.VertexPixels.Sum(x => x.Item1.z) /
p2.VertexPixels.Count;
                return
avgZ2.CompareTo(avgZ1);
            });

            foreach (var polygon in
_projections)
            {
                foreach (var point in
polygon.Pixels)
```

MatrixManager.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using UnityEngine;

namespace Assets.Scripts
{
    public static class MatrixManager
    {
        public static double[,]
Multiplicate(double[,] matrixA, double[,]
matrixB)
        {
            int sizeN =
matrixA.GetUpperBound(0) + 1;
            int sizeM =
matrixA.GetUpperBound(1) + 1;
            int sizeK =
matrixB.GetUpperBound(1) + 1;
            if (sizeM !=
matrixB.GetUpperBound(0) + 1)
                throw new Exception("Columns
count of the matrix A must be equal to rows
count of the matrix B!");

            double[,] result = new
double[sizeN, sizeK];

            for (int i = 0; i < sizeN; i++)
                for (int j = 0; j < sizeK;
j++)
                {
                    double value = 0;
                    for (int z = 0; z <
sizeM; z++)
                        value += matrixA[i,
z] * matrixB[z, j];
                    result[i, j] = value;
                }

            return result;
        }
                {
                    int x =
(int)point.Item1.x;
                    int y =
(int)point.Item1.y;
                    colorBuffer[x, y] =
point.Item2;
                }
            }

            return colorBuffer;
        }
    }
}
```

```
        public static double[,]
Divide(double[,] matrix, double value)
        {
            int sizeN =
matrix.GetUpperBound(0) + 1;
            int sizeM =
matrix.GetUpperBound(1) + 1;

            double[,] result = new
double[sizeN, sizeM];

            for (int i = 0; i < sizeN; i++)
                for (int j = 0; j < sizeM;
j++)
                {
                    result[i, j] = matrix[i,
j] / value;
                }

            return result;
        }

        public static double[,]
GetPerspectiveMatrix(double near, double far,
double angleOfView, Vector2 size)
        {
            double[,] result = new double[4,
4];

            // set the basic projection
matrix
            double scaleY = 1 /
Math.Tan(angleOfView * 0.5 * Math.PI / 180);
            double scaleX = scaleY / (size.x
/ size.y);
            result[0, 0] = scaleX; // scale
the x coordinates of the projected point
            result[1, 1] = scaleY; // scale
the y coordinates of the projected point
            result[2, 2] = -far / (far -
near); // used to remap z to [0,1]
            result[3, 2] = -far * near / (far
- near); // used to remap z [0,1]
            result[2, 3] = -1; // set w = -z
            result[3, 3] = 0;

            return result;
        }
    }
```

```
}
```

ProjectionManager.cs

```
using Assets.Scripts.Models;
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using UnityEngine;
using UnityEngine.UIElements;

namespace Assets.Scripts
{
    public static class ProjectionManager
    {
        public static TriangleProjection
ProjectVertexes(TriangleProjection proj, int
width, int height)
        {
            foreach (var point in
proj.Points)
            {
                int x = Mathf.Min(width - 1,
(int)((point.x + 1) * 0.5 * width));
                int y = Mathf.Min(height - 1,
(int)((1 - (point.y + 1) * 0.5) * height));
                proj.VertexPixels.Add(new
Tuple<Vector3, Color>(new Vector3(x, y,
point.z), Color.black));
            }
            return proj;
        }

        public static TriangleProjection
ProjectEdges(TriangleProjection proj)
        {
            var PointA = proj.Pixels[0];
            var PointB = proj.Pixels[1];
            var PointC = proj.Pixels[2];

            var lineAB =
FindLineBetweenPoints(new
Vector2Int((int)PointA.Item1.x,
(int)PointA.Item1.y), new
Vector2Int((int)PointB.Item1.x,
(int)PointB.Item1.y));
            foreach (var point in lineAB)
            {
                var z =
DefineZBetweenTwoPoints(point, PointA.Item1,
PointB.Item1);
                proj.EdgeABPixels.Add(new
Tuple<Vector3, Color>(new Vector3(point.x,
point.y, z), Color.black));
            }

            var lineBC =
FindLineBetweenPoints(new
Vector2Int((int)PointB.Item1.x,
(int)PointB.Item1.y), new
Vector2Int((int)PointC.Item1.x,
(int)PointC.Item1.y));
            foreach (var point in lineBC)
            {
                var z =
DefineZBetweenTwoPoints(point, PointB.Item1,
PointC.Item1);
                proj.EdgeBCPixels.Add(new
Tuple<Vector3, Color>(new Vector3(point.x,
point.y, z), Color.black));
            }

            var lineAC =
FindLineBetweenPoints(new
Vector2Int((int)PointA.Item1.x,
(int)PointA.Item1.y), new
Vector2Int((int)PointC.Item1.x,
(int)PointC.Item1.y));
            foreach (var point in lineAC)
            {
                var z =
DefineZBetweenTwoPoints(point, PointA.Item1,
PointC.Item1);
                proj.EdgeACPixels.Add(new
Tuple<Vector3, Color>(new Vector3(point.x,
point.y, z), Color.black));
            }

            return proj;
        }

        public static TriangleProjection
ProjectInnerPoints(TriangleProjection proj)
        {
            var PointA =
proj.VertexPixels[0];
            var PointB =
proj.VertexPixels[1];
            var PointC =
proj.VertexPixels[2];

            var minX =
Mathf.Min(PointA.Item1.x, PointB.Item1.x,
PointC.Item1.x);
            var maxX =
Mathf.Max(PointA.Item1.x, PointB.Item1.x,
PointC.Item1.x);

            List<int> minPointNums = new
List<int>();
            List<int> maxPointNums = new
List<int>();
            int midPointNum = -1;
            for (int i = 0; i <
proj.VertexPixels.Count; i++)
            {
                if
(proj.VertexPixels[i].Item1.x == minX)
                    minPointNums.Add(i);

                if
(proj.VertexPixels[i].Item1.x == maxX)
                    maxPointNums.Add(i);
```

```
                if
(!(proj.VertexPixels[i].Item1.x == minX ||
proj.VertexPixels[i].Item1.x == maxX))
                    midPointNum = i;
            }

            if (midPointNum == -1)
            {
                if (minPointNums.Count > 1)
                {
                    midPointNum =
minPointNums[1];
                }
                else if (maxPointNums.Count >
1)
                {
                    midPointNum =
maxPointNums[1];
                }
            }

            List<Vector3> longestEdge;
            List<Vector3> shortEdges;
            switch (midPointNum)
            {
                case 0:
                    longestEdge =
proj.EdgeBCPixels.Select(x =>
x.Item1).ToList();
                    shortEdges =
proj.EdgeACPixels.Select(x =>
x.Item1).ToList();

shortEdges.AddRange(proj.EdgeABPixels.Select(
x => x.Item1).ToList());
                    break;
                case 1:
                    longestEdge =
proj.EdgeACPixels.Select(x =>
x.Item1).ToList();
                    shortEdges =
proj.EdgeBCPixels.Select(x =>
x.Item1).ToList();

shortEdges.AddRange(proj.EdgeABPixels.Select(
x => x.Item1).ToList());
                    break;
                case 2:
                    longestEdge =
proj.EdgeABPixels.Select(x =>
x.Item1).ToList();
                    shortEdges =
proj.EdgeBCPixels.Select(x =>
x.Item1).ToList();

shortEdges.AddRange(proj.EdgeACPixels.Select(
x => x.Item1).ToList());
                    break;
                default:
                    throw new Exception("Mid
point cannot be NULL!");
            }

            bool longestEdgeIsHigher =
proj.VertexPixels[midPointNum].Item1.y <
longestEdge.Where(p => p.x ==
proj.VertexPixels[midPointNum].Item1.x).Avera
ge(p => p.y);

            for (int i = (int)minX + 1; i <
maxX - 1; i++)
            {
                int minY, maxY;
                float minZ, maxZ;

                if (!longestEdgeIsHigher)
                {
                    var tmpMin =
longestEdge.Where(p => p.x == i).ToList();
                    var tmpMax =
shortEdges.Where(p => p.x == i).ToList();

                    if (tmpMin.Count == 0)
                        tmpMin =
longestEdge.Where(p => p.x == i - 1 || p.x ==
i + 1).ToList();
                    if (tmpMax.Count == 0)
                        tmpMax =
shortEdges.Where(p => p.x == i - 1 || p.x ==
i + 1).ToList();

                    minY = tmpMin.Max(p =>
(int)p.y);
                    minZ = tmpMin.Where(p =>
p.y == minY).First().z;
                    maxY = tmpMax.Min(p =>
(int)p.y);
                    maxZ = tmpMax.Where(p =>
p.y == maxY).First().z;
                }
                else
                {
                    var tmpMin =
shortEdges.Where(p => p.x == i).ToList();
                    var tmpMax =
longestEdge.Where(p => p.x == i).ToList();

                    if (tmpMin.Count == 0)
                        tmpMin =
shortEdges.Where(p => p.x == i - 1 || p.x ==
i + 1).ToList();
                    if (tmpMax.Count == 0)
                        tmpMax =
longestEdge.Where(p => p.x == i - 1 || p.x ==
i + 1).ToList();

                    minY = tmpMin.Max(p =>
(int)p.y);
                    minZ = tmpMin.Where(p =>
p.y == minY).First().z;
                    maxY = tmpMax.Min(p =>
(int)p.y);
                    maxZ = tmpMax.Where(p =>
p.y == maxY).First().z;
                }
```

```
                for (int j = minY + 1; j <
maxY - 1; j++)
                {
                    var z =
DefineZBetweenTwoPoints(new Vector2Int(i, j),
new Vector3(i, minY, minZ), new Vector3(i,
maxY, maxZ));
                    proj.InnerPixels.Add(new
Tuple<Vector3, Color>(new Vector3(i, j, z),
Color.white));
                }
            }

            return proj;
        }

        private static List<Vector2Int>
FindLineBetweenPoints(Vector2Int p1,
Vector2Int p2)
        {
            List<Vector2Int> line = new
List<Vector2Int>();

            if (p1 == p2)
                line.Add(new Vector2Int(p1.x,
p1.y));

            Vector2 t = p1;
            double frac = 1 /
Mathf.Sqrt(Mathf.Pow(p2.x - p1.x, 2) +
Mathf.Pow(p2.y - p1.y, 2));
            double ctr = 0;

            while ((int)t.x != p2.x ||
(int)t.y != p2.y)
            {
                t = Vector2.Lerp(p1, p2,
(float)ctr);
                ctr += frac;
                line.Add(new
Vector2Int((int)t.x, (int)t.y));
            }

            return line;
        }

        private static float
DefineZBetweenTwoPoints(Vector2Int p, Vector3
p1, Vector3 p2)
        {
            var distance1 =
Vector2Int.Distance(p, new
Vector2Int((int)p1.x, (int)p1.y));
            var distance2 =
Vector2Int.Distance(p, new
Vector2Int((int)p2.x, (int)p2.y));
            if (distance1 == 0 && distance2
== 0)
                return p1.z;

            var z = (distance1 / (distance1 +
distance2)) * (p2.z - p1.z) + p1.z;
            return z;
        }
    }
}
```

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
МІНІСТЕРСТВО ІНФРАСТРУКТУРИ УКРАЇНИ
ДНІПРОВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ
ЗАЛІЗНИЧНОГО ТРАНСПОРТУ
імені академіка В. Лазаряна

ТЕЗИ
XIV-ої Міжнародної науково-практичної конференції «СУЧАСНІ ІНФОРМАЦІЙНІ І КОМУНІКАЦІЙНІ ТЕХНОЛОГІЇ НА ТРАНСПОРТІ, В ПРОМИСЛОВОСТІ ТА ОСВІТІ»
15-16 грудня 2020

Тезисы
XIV-й Международной научно-практической конференции «СОВРЕМЕННЫЕ ИНФОРМАЦИОННЫЕ И КОММУНИКАЦИОННЫЕ ТЕХНОЛОГИИ НА ТРАНСПОРТЕ, В ПРОМЫШЛЕННОСТИ И ОБРАЗОВАНИИ»
15-16 декабря 2020

ABSTRACTS
of the XIV-th International Conference «MODERN INFORMATION AND COMMUNICATION TECHNOLOGIES ON A TRANSPORT, IN INDUSTRY AND EDUCATION»
15-16, December, 2020

Міністерство освіти і науки України

Дніпровський національний університет
залізничного транспорту ім. акад. В. Лазаряна

Східний науковий центр транспортної академії наук





TEMPUS: CITISET & SEREIN & CRENG

ТЕЗИ
XIV Міжнародної науково-практичної конференції
«СУЧАСНІ ІНФОРМАЦІЙНІ ТА КОМУНІКАЦІЙНІ ТЕХНОЛОГІЇ НА ТРАНСПОРТІ, В ПРОМИСЛОВОСТІ ТА ОСВІТІ»

ABSTRACTS
of the XIV International Conference
«MODERN INFORMATION AND COMMUNICATION TECHNOLOGIES ON A TRANSPORT, IN INDUSTRY AND EDUCATION»

ТЕЗИСЫ
XIV Международной научно-практической конференции
«СОВРЕМЕННЫЕ ИНФОРМАЦИОННЫЕ И КОММУНИКАЦИОННЫЕ ТЕХНОЛОГИИ НА ТРАНСПОРТЕ, В ПРОМЫШЛЕННОСТИ И ОБРАЗОВАНИИ»

15.12.2020 – 16.12.2020

Дніпро
2020

УДК 658.512.2:681.3.06

Сучасні інформаційні та комунікаційні технології на транспорті, в промисловості і освіті: Тези XIV Міжнародної науково-практичної конференції (Дніпро, 15-16 грудня 2020 р.). – Д .: ДІІТ, 2020. – 159 с.

У збірнику представлені тези доповідей XIV Міжнародної науково-практичної конференції «Сучасні інформаційні та комунікаційні технології на транспорті, в промисловості та освіті», яка відбулася 15-16 грудня 2020 року в Дніпровському національному університеті залізничного транспорту імені академіка В. Лазаряна в онлайн форматі. Розглянуто результати теоретичних і експериментальних досліджень, а також проблемні питання функціонування та перспективи розвитку інформаційних технологій транспорту, промисловості й освіти.
Збірник призначений для науково-технічних працівників залізниць, підприємств транспорту, викладачів вищих навчальних закладів, докторантів, аспірантів і студентів.

Адреса редакційної колегії:
49010, м. Дніпро, вул. Лазаряна, 2, ДНУЗТ

Тези доповідей друкуються мовою оригіналу в редакції авторів.

ДОСЛІДЖЕННЯ ЧАСОВОЇ ЕФЕКТИВНОСТІ АЛГОРИТМІВ ВИДАЛЕННЯ НЕВИДИМИХ ЛІНІЙ ТА ПОВЕРХОНЬ У КОМП'ЮТЕРНІЙ ГРАФІЦІ

Автор – Строчіков І.О., студент групи ПЗ1921
Науковий керівник – к.т.н., доц. Іванов О.П.
Дніпровський національний університет імені академіка В. Лазаряна
Україна

В наш час вже більшість людей так чи інакше стикаються з комп'ютерною графікою, бо у кожного є власний смартфон або персональний комп'ютер. Для комфортного використання програм для моделювання або ігор з комп'ютерною графікою дуже важливо, аби тривимірні об'єкти на сцені відмальовувались максимально швидко з прийнятною частотою (40-60 кадрів в секунду).

Тому варто звернути увагу на ефективність алгоритмів видалення невидимих ліній та поверхонь, бо вони займають доволі велику частку загального часу рендеру тривимірних об'єктів. Ці алгоритми вирішують проблему видалення тих ділянок тривимірного об'єкту, які приховані за іншими поверхнями та мають бути невидимими з точки зору спостерігача. Алгоритми видалення невидимих ліній та поверхонь можуть працювати у просторі екрану або об'єктів. В залежності від цього їх поділяють на відповідні категорії. До працюючих у просторі екрану відносяться алгоритми Z-буферу, scan-line, «художника», Варнока та інші. Представників другої категорії набагато менше, серед відомих до них відноситься лише алгоритм Робертса.

Але на практиці використовуються лише деякі з них. Це ті алгоритми, які показують найкращі результати для більшості ситуацій. Наприклад, в широко відомій графічній бібліотеці OpenGL реалізовано лише алгоритм Z-буфера. І користувач бібліотеки зможе використати лише цей алгоритм для видалення невидимих ліній та поверхонь. І хоча цей алгоритм зможе опрацювати тривимірну графічну сцену абсолютно будь-якої складності, це ще не означає, що він є найефективнішим в будь-якому сценарії.

До недавніх часів це не було серйозною проблемою, бо навіть якщо обраний алгоритм буде працювати трохи повільніше іншого, то різниця буде настільки малою, що користувач просто цього не помітить, а частота кадрів в секунду буде все ще прийнятною. Але зараз з'явилися мобільні гаджети, які мають набагато меншу обчислювальну потужність, аніж персональні комп'ютери. Тим паче вони мають меншу кількість оперативної пам'яті, що може позначитися на ефективності алгоритмів, яким для роботи якраз потрібно багато пам'яті.

З розширенням технічних можливостей пристроїв, виросли й вимоги до комп'ютерної графіки. Користувачі почали звикати до поточного стану, коли комп'ютерна тривимірна графіка, що відображається, виглядає дуже реалістичною за рахунок більш різноманітної палітри кольорів, плавної анімації та високої деталізації. Але з появою мобільних пристроїв виявилося, що досягти такого ж результату й тут неможливо, тому розробникам мобільних додатків та ігор часто доводиться зменшувати якість комп'ютерної графіки або витрачати велику кількість часу та ресурсів на оптимізацію задля більшої продуктивності.

Інші дослідження щодо алгоритмів видалення невидимих ліній та поверхонь зосереджують увагу на загальному порівнянні алгоритмів, визначенні їх сильних та слабких сторін. Тому у цьому дослідженні буде додатково визначено рівень впливу основних показників (роздільна здатність екрану, кількість полігонів, кількість пікселів для обробки алгоритмом) на час роботи алгоритмів. На основі результатів цього дослідження інші розробники зможуть розробити інструмент, який буде розраховувати очікуваний час роботи алгоритмів видалення невидимих ліній та поверхонь та обирати найефективніший з них для конкретної ситуації.

ДОСЛІДЖЕННЯ ЧАСОВОЇ ЕФЕКТИВНОСТІ АЛГОРИТМІВ ВИДАЛЕННЯ НЕВИДИМИХ ЛІНІЙ ТА ПОВЕРХОНЬ У КОМП'ЮТЕРНІЙ ГРАФІЦІ

Автор: Строчіков І.О.

Науковий керівник – к.т.н., доц. Іванов О.П.

Дніпровський національний університет імені академіка В. Лазаряна

ВСТУП

В наш час вже більшість людей так чи інакше стикаються з комп'ютерною графікою, бо у кожного є власний смартфон або персональний комп'ютер. Для комфортного використання програм для моделювання або ігор з комп'ютерною графікою дуже важливо, аби тривимірні об'єкти на сцені відмальовувались максимально швидко з прийнятною частотою (40-60 кадрів в секунду).

Тому варто звернути увагу на ефективність алгоритмів видалення невидимих ліній та поверхонь, бо вони займають доволі велику частку загального часу рендеру тривимірних об'єктів. Ці алгоритми вирішують проблему видалення тих ділянок тривимірного об'єкту, які приховані за іншими поверхнями та мають бути невидимими з точки зору спостерігача. Алгоритми видалення невидимих ліній та поверхонь можуть працювати у просторі екрану або об'єктів. В залежності від цього їх поділяють на відповідні категорії. До працюючих у просторі екрану відносяться алгоритми Z-буферу, scan-line, «художника», Варнока та інші. Представників другої категорії набагато менше, серед відомих до них відноситься лише алгоритм Робертса.

Але на практиці використовуються лише деякі з них. Це ті алгоритми, які показують найкращі результати для більшості ситуацій. Наприклад, в широко відомій графічній бібліотеці OpenGL реалізовано лише алгоритм Z-буфера. І користувач бібліотеки зможе використати лише цей алгоритм для видалення невидимих ліній та поверхонь. І хоча цей алгоритм зможе опрацювати тривимірну графічну сцену абсолютно будь-якої складності, це ще не означає, що він є найефективнішим в будь-якому сценарії.

До недавніх часів це не було серйозною проблемою, бо навіть якщо обраний алгоритм буде працювати трохи повільніше іншого, то різниця буде настільки малою, що користувач просто цього не помітить, а частота кадрів в секунду буде все ще прийнятною. Але зараз з'явилися мобільні гаджети, які мають набагато меншу обчислювальну потужність, аніж персональні комп'ютери. Тим паче вони мають меншу кількість оперативної пам'яті, що може позначитися на ефективності алгоритмів, яким для роботи якраз потрібно багато пам'яті.

З розширенням технічних можливостей пристроїв, виросли й вимоги до комп'ютерної графіки. Користувачі почали звикати до

поточного стану, коли графіка, що відображається, виглядає дуже реалістичною за рахунок більш різноманітної палітри кольорів та плавної анімації. Але з появою мобільних пристроїв виявилося, що досягти такого ж результату й тут неможливо, тому розробникам мобільних додатків та ігор часто доводиться зменшувати якість комп'ютерної графіки або витрачати велику кількість часу та ресурсів на оптимізацію задля більшої продуктивності.

ПОСТАНОВКА ЗАДАЧІ

Розробити програмний продукт, який можна буде застосувати для проведення експериментів з алгоритмами видалення невидимих ліній та поверхонь та зібрати необхідні дані. На основі зібраних даних проаналізувати та визначати рівень впливу кожного показника на час роботи HSR-алгоритмів.

МЕТОДИКА ДОСЛІДЖЕННЯ

Часова ефективність кожного з досліджуваних алгоритмів буде обчислюватися за результатами експериментів. Дуже важливо виконати експеримент декілька сотень разів, щоб більш точно встановити середній показник часу. Середній показник часу для одного експерименту буде обчислюватися за такою формулою:

$$T_{avg} = \frac{\sum_{i=0}^{n} t_i}{n}, \qquad (1)$$

де Tavg – середній час роботи алгоритму при заданих вхідних даних,

n – кількість експериментів,

ti – час роботи алгоритму в і-тому експерименті.

Слід зауважити, що аномальні результати експериментів будуть виключені з загального підрахунку.

Після того, як будуть знайдені показники середнього часу для декількох різних експериментів, по цим даним буде побудований графік та лінія тренду до нього. За допомогою засобів програми Excel можна буде знайти функцію побудованої лінії тренду, наприклад, такого виду: y = 0.7*x + 53.2, якщо залежність буде лінійною. Підставляючи деяку характеристику (наприклад, роздільну здатність екрану) під x, можна буде обчислити приблизний час роботи алгоритму.

На основі обчисленого приблизного часу роботи HSR-алгоритму можна буде вирішити, який з них краще використати у конкретній ситуації.

Для дослідження часової ефективності алгоритмів видалення невидимих ліній та поверхонь були обрані різні тривимірні моделі для кожного з експериментів. Для одного з експериментів було вирішено використовувати модель кулі з різним ступенем деталізації. Це можна побачити на рис. 1. Мінімальний рівень деталізації – 80 полігонів, максимальний – 39 тисяч полігонів.

Фактично це одна й та сама модель, тому кількість пікселів для обробки буде зростати незначно зі зростанням кількості полігонів. Це необхідно для того, щоб виявити залежність часової ефективності алгоритму саме від кількості полігонів.

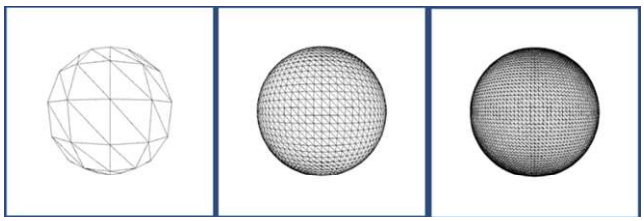

Рисунок 1 – Модель кулі з різним степенем деталізації

Для іншого експерименту була обрана дуже проста модель площини, яка містить всього 8 полігонів. Це було зроблено для того, щоб виключити залежність алгоритму від кількості полігонів моделі. Зображення цієї моделі на рис. 2.

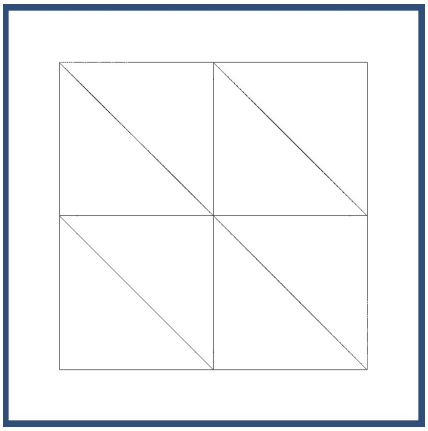

Рисунок 2 – Модель площини з 8 полігонами

Для останнього експерименту було обрано модель чайника середньої складності з 10 тисячами полігонів. Він буде використаний для експерименту з дослідження залежності часової ефективності алгоритму від роздільної здатності екрану. Зображення цієї моделі на рис. 3.

Рисунок 3 – Тривимірна модель чайника

ПРОГРАМНЕ ЗАБЕЗПЕЧЕННЯ ДЛЯ ДОСЛІДЖЕННЯ

Для цього дослідження було спроектоване та розроблене програмне забезпечення, яке дозволяє швидко збирати дані часу роботи HSR-алгоритмів при заданих вхідних параметрах.

На екрані налаштування (рис. 4) користувач може обрати висоту та ширину зображення, один з HSR-алгоритмів (Z-буфера, scan-line або «художника») для обробки, кількість циклів запуску та файл з тривимірною моделлю.

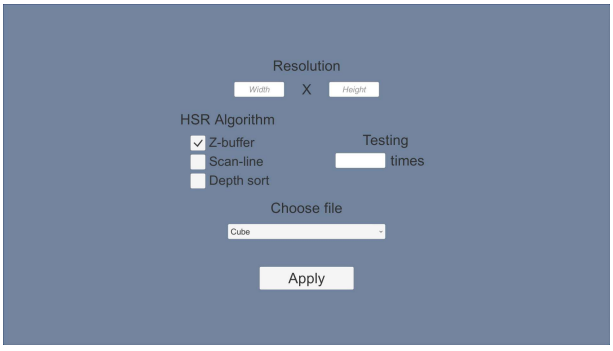

Рисунок 4 – Екран налаштування

Після цього з’являється екран з візуальним представленням обраної тривимірної моделі (рис. 5), яка вже була

оброблена обраним HSR-алгоритмом і не містить невидимих ліній та поверхонь.

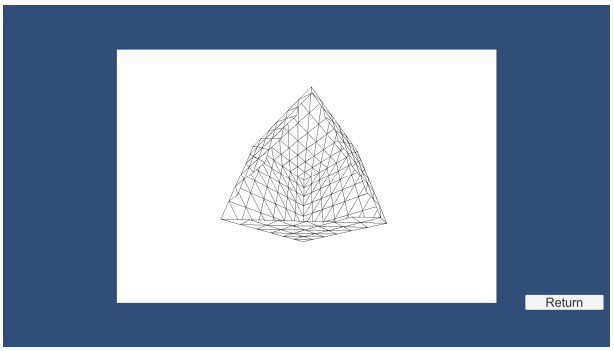

Рисунок 5 – Екран з тривимірною моделлю

Але найголовніше – це те, що в окремий файл формату .csv записуються результати цього експерименту. Дані в цьому файлі представлені в зручному вигляді для подальшого аналізу.

РЕЗУЛЬТАТИ ДОСЛІДЖЕННЯ

Весь експеримент можна поділити на три етапи:

- визначення залежності від кількості полігонів;
- визначення залежності від кількості пікселів для обробки;
- визначення залежності від роздільної здатності екрану.

На основі отриманих даних був побудований графік, який візуально відображає залежність часу роботи від кількості пікселів для обробки для кожного алгоритму. Також були побудовані лінії тренду до кожної лінії, а також були визначені функції цих ліній тренду. Це дозволяє зрозуміти, наскільки змінюється час алгоритму від того чи іншого показника. Побудований графік з лініями тренду зображено на рис. 6.

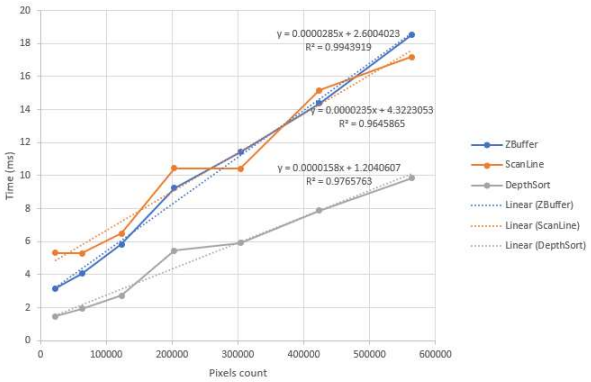

Рисунок 6 – Залежність часу роботи від кількості пікселів до обробки

Визначені функції для кожної лінії тренду показують на те, як саме змінюється час роботи алгоритму в залежності від кількості пікселів.

Найменшу залежність від цього показника демонструє алгоритм «художника». В середньому час роботи цього алгоритму підвищується на 1.58 мс з кожними 100 тисяч пікселів.

Наступним по порядку йде scan-line алгоритм, час роботи якого підвищується на 2.35 мс за кожні 100 тисяч нових пікселів.

І найбільш залежним від цього показнка виявився алгоритм Z-буфера, який має результат в 2.85 мс за кожні 100 тисяч пікселів.

На рис. 7 зображено графік залежності часу роботи алгоритмів лише від кількості полігонів.

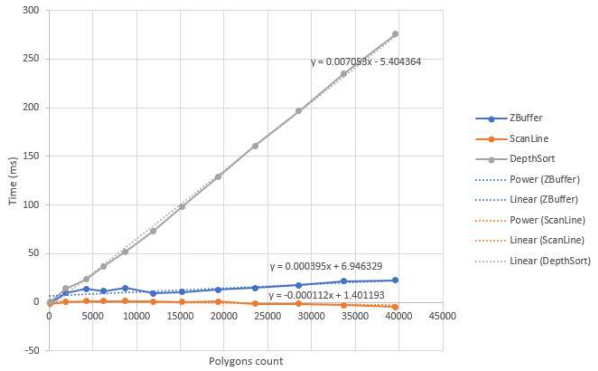

Рисунок 7 – Залежність часу роботи від кількості полігонів моделі

З графіку видно, що час алгоритму «художника» дуже залежить від кількості полігонів. Лінія тренду scan-line алгоритму направлена вниз, що свідчить про те, що цей алгоритм взагалі не залежить від кількості полігонів. Лінія тренду має бути рівною, а коефіцієнт впливу дорівнювати нулю, а не від'ємному числу. Скоріше за все так сталося через деяку похибку при вимірюванні, бо коефіцієнт впливу наближений до нуля. Що стосується алгоритму Z-буфера, то він все ж таки залежить від зміни кількості полігонів, хоча й зовсім трохи.

Загалом отримано такі коефіцієнти впливу:

– час роботи алгоритму «художника» збільшується на 70.53 мс з кожними 10 тисяч полігонів;

– час роботи scan-line алгоритму не змінюється взагалі, тобто коефіцієнт зміни дорівнює 0;

– час роботи алгоритму Z-буферу збільшується на 3.95 мс з кожними 10 тисяч полігонів.

Також було побудовано ще один графік, який зображено на рис. 8, який відображає залежність часу роботи лише від роздільної здатності екрану.

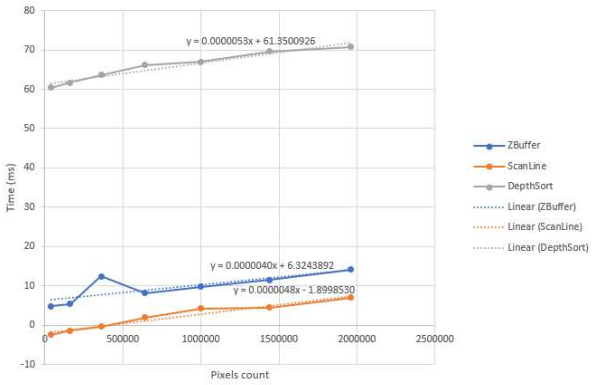

Рисунок 7 – Залежність часу роботи від роздільної здатності екрану

З графіку видно, що кожен з алгоритмів зовсім трохи залежить від роздільної здатності екрану. Менше за всіх від цього показника залежить алгоритм Z-буфера, а більше – алгоритм «художника». Але різниця між ними настільки мала, що майже не грає ролі.

Загалом отримано такі коефіцієнти впливу:

– час роботи алгоритму «художника» збільшується на 5.3 мс з кожним мільйоном пікселів;

– час роботи scan-line алгоритму збільшується на 4.8 мс з кожним мільйоном пікселів;

– час роботи алгоритму Z-буферу збільшується на 4 мс з кожним мільйоном пікселів.

Отже, було визначено коефіцієнти впливу кожного показника на досліджувані алгоритми видалення невидимих ліній та поверхонь. Підсумкова таблиця зображена нижче (рис. 8).

Алгоритм	Коеф. впливу кількості пікселів до обробки	Коеф. впливу кількості полігонів	Коеф. впливу роздільної здатності екрану
Z-буфера	0.0000285	0.000395	0.000004
Scan-line	0.0000235	0	0.0000048
«Художника»	0.0000158	0.007053	0.0000053

Рисунок 8 – Підсумкова таблиця з коефіцієнтами впливу

Тепер маючи ці коефіцієнти можна обчислювати приблизний час роботи алгоритму в залежності від вхідних даних за формулою 2:

$$T_e = K_{pix} * Pix + K_{pol} * Pol + K_{Res} * * H * W, \quad (2)$$

де T_e – приблизний час роботи алгоритму,

K_{pix} – коефіцієнт впливу кількості пікселів до обробки на час роботи,

K_{pol} – коефіцієнт впливу кількості полігонів на час роботи,

K_{res} – коефіцієнт впливу кількості роздільної здатності екрану на час роботи,

Pix – кількість пікселів до обробки,

Pol – кількість полігонів,

H – висота зображення,

W – ширина зображення.

Було визначено, що в цілому обчислений час роботи більш-менш співпадає з фактичними результатами з приблизним відхиленням у 15-20%.

ВИСНОВКИ

Після аналізу зібраної інформації було з’ясовано, що на час роботи кожного з HSR-алгоритмів одні й ті самі показники впливають по-різному. Наприклад, кількість полігонів дуже впливає на час роботи алгоритму «художника» та зовсім не впливає на час роботи scan-line алгоритму. Щодо роздільної здатності екрану – цей показник впливає на час роботи усіх досліджуваних HSR-алгоритмів приблизно однаково. Показник «кількість пікселів до обробки» найменше впливає на час роботи алгоритму «художника» і майже в два рази більше – на алгоритм Z-буферу.

На основі отриманих даних було розраховано коефіцієнти впливу (див. рис. 8) для кожного показника, які числено відображають ступінь впливу на час роботи кожного алгоритму. Знайдені коефіцієнти впливу можна застосувати при реалізації методу-перемикача, який буде обирати найефективніший HSR-алгоритм в залежності від вхідних даних. Він буде обчислювати очікуваний час кожного з алгоритмів, а потім обирати той, що отримав найкращий результат.